DISCRETIZATION AND APPROXIMATION METHODS FOR REINFORCEMENT LEARNING OF HIGHLY RECONFIGURABLE SYSTEMS

A Dissertation

by

AMANDA KATHRYN LAMPTON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2009

Major Subject: Aerospace Engineering

DISCRETIZATION AND APPROXIMATION METHODS FOR REINFORCEMENT LEARNING OF HIGHLY RECONFIGURABLE SYSTEMS

A Dissertation

by

AMANDA KATHRYN LAMPTON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	John Valasek
Committee Members,	John Junkins
	Suman Chakravorty
	Thomas Ioerger
Head of Department,	Dimitri Lagoudas

December 2009

Major Subject: Aerospace Engineering

ABSTRACT

Discretization and Approximation Methods for
Reinforcement Learning of Highly Reconfigurable Systems. (December 2009)
Amanda Kathryn Lampton, B.S., Texas A&M University;
M.S., Texas A&M University
Chair of Advisory Committee: Dr. John Valasek

There are a number of techniques that are used to solve reinforcement learning problems, but very few that have been developed for and tested on highly reconfigurable systems cast as reinforcement learning problems. Reconfigurable systems refers to a vehicle (air, ground, or water) or collection of vehicles that can change its geometrical features, i.e. shape or formation, to perform tasks that the vehicle could not otherwise accomplish. These systems tend to be optimized for several operating conditions, and then controllers are designed to reconfigure the system from one operating condition to another. Q-learning, an unsupervised episodic learning technique that solves the reinforcement learning problem, is an attractive control methodology for reconfigurable systems. It has been successfully applied to a myriad of control problems, and there are a number of variations that were developed to avoid or alleviate some limitations in earlier version of this approach. This dissertation describes the development of three modular enhancements to the Q-learning algorithm that solve some of the unique problems that arise when working with this class of systems, such as the complex interaction of reconfigurable parameters and computationally intensive models of the systems. A multi-resolution state-space discretization method is developed that adaptively rediscretizes the state-space by progressively finer grids around one or more distinct Regions Of Interest within the state or learning space. A genetic algorithm that autonomously selects the basis functions to be used in the approxi-

mation of the action-value function is applied periodically throughout the learning process. Policy comparison is added to monitor the state of the policy encoded in the action-value function to prevent unnecessary episodes at each level of discretization. This approach is validated on several problems including an inverted pendulum, reconfigurable airfoil, and reconfigurable wing. Results show that the multi-resolution state-space discretization method reduces the number of state-action pairs, often by an order of magnitude, required to achieve a specific goal and the policy comparison prevents unnecessary episodes once the policy has converged to a usable policy. Results also show that the genetic algorithm is a promising candidate for the selection of basis functions for function approximation of the action-value function.

To my parents and sister, for their unwavering support and belief in me

ACKNOWLEDGMENTS

This has been a long and arduous process filled with both good times and challenging times. From the first days of my undergraduate degree, through the final push to finish this doctoral work, many of the worst battles have been internal ones. As such, my humblest and sincerest gratitude goes to those people who have supported me emotionally and offered guidance when I felt lost.

I consider myself incredibly lucky to be the child of my parents. They have never wavered in their support of my dreams. My mother has always stood by me to offer support when it felt like the world was crashing in around me, to calm me and offer guidance when problems seemed too hard to solve, and celebrate with me when I made progress in life or research. My father is an ever-present quiet strength that I have sought innumerable times during this process. A smile, a lively conversation, a shared experience such as hiking to a green sand beach or an aimless drive through the Texas hill country with my father have taught me much about what it is to live in and enjoy this world.

Though she has not had as much of a physical presence in my life of late, I can always rely on the accepting and constant friendship of Sarah Townsend. The glowing support from her has often rallied my flagging spirits when I am particularly down. My first friend in college, Alicia Rutledge, has always been good for commiserating over classes, random coffee nights, girls' night out, and the occasional trip to Hawai'i. My undergraduate days would have a been a lot darker without her friendship. Though she pursues her doctoral studies at a different university, she is always good for a lively phone conversation or email exchange. I offer particular thanks to my undergraduate roommate, Magdalena Gotsch, and my current roommate, Kathryn Stribley. During my undergraduate days, they both fought hard to ensure that I

had some semblance of a social life. Rock climbing on Fridays and joining our dorm for dinner did much to preserve my sanity and well-being in those days and taught me the value of maintaining a balanced life during my graduate career and beyond. Kathryn's continued companionship during my graduate career offered a warm living environment and a welcomed escape from the challenges of technical research.

Two others I would like to mention personally are my sister, Michelle Lampton, and Mark Stonger. I consider Michelle one of my closest and best friends. The last several years have brought us closer together, and much of what I have accomplished here would have been much more painful without her. I thank her for being my friend, confidante, staunch supporter, workout partner, and partner in mischief. The love and support offered by Mark over the last two years has been immeasurable. He has been a friend to talk to, a shoulder to cry on, a person to celebrate with, and a companion to share an evening with.

I would like to acknowledge and thank the many people who contributed to the successful completion of this dissertation. Most notably Dr. John Valasek, who has been an exceptional advisor and mentor throughout my graduate career and through much of my undergraduate career. I consider myself extremely lucky to have been able to work for and alongside him. The guidance of my committee in the technical aspects of this dissertation is greatly appreciated. I would like to thank Dr. Thomas Ioerger whose guidance has been invaluable to my understanding of artificial intelligence and the computer scientist approach to machine learning. I am thankful to Dr. Suman Chakravorty for his time and expertise offered in numerous group meetings. I would also like to thank Dr. John Junkins for his aid in my understanding of estimation theory, dynamic systems, and control theory. I am very thankful to Adam Niksch for developing the airfoil and wing computational fluid dynamic models without which this dissertation would not be possible.

Finally, I am most grateful to the rest of my family, whose love and support made everything possible.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1

A. Introduction to the Class of Systems 1
B. Literature Review 2
1. Reconfigurable Air Vehicle Literature Review 2
a. Design of Highly Reconfigurable Systems 3
b. Wing Reconfiguration 5
c. Structure of Reconfigurable Systems 6
d. Performance, Simulation, Control, and Testing of Highly Reconfigurable Systems 7
e. Morphing Airfoil 9
f. Biologically Inspired Morphing Aircraft 10
2. Reinforcement Learning Literature Review 11
a. Dynamic Programming 12
b. Iterative Learning Control 14
c. Fuzzy Q-Learning 15
d. Methods Related to Q-Learning 15
e. Multiagent and Hierarchical Reinforcement Learning 16
f. Learning on a Continuous Domain 18
g. Applications of Machine Learning Algorithms . . 20
h. Learning Applied to Morphing 21
C. Research Objectives/Motivation 22
D. Scope 24
E. Contribution to Body of Knowledge 24
F. Organization 29

II REINFORCEMENT LEARNING 30

A. Introduction to Reinforcement Learning 30
B. Derivation of Q-Learning Algorithm 35
C. Exploration vs. Exploitation 37
D. Proof of Convergence for Q-Learning 39
1. The Action-Replay Process 40
2. Convergence of Q^*_{ARP} to Q^* 43

CHAPTER Page

III STATE-SPACE DISCRETIZATION OF A CONTINUOUS DOMAIN FOR Q-LEARNING 45

A. Problem Definition 45
B. Learning on a 2-, 4-, and N-Dimensional Domain 45
C. Multi-Resolution State-Space Discretization (AAG) for N-Dimensions 50
D. Multiple Goal Regions 53

IV FUNCTION APPROXIMATION 55

A. Introduction to Genetic Algorithms 55
B. Genetic Algorithm for Function Approximation 59
1. Basis Function Selection 62
2. Fitness Calculation 65
3. Integration in Q-learning Algorithm 66
C. Illustrative Examples 67
1. Simple Polynomial 67
a. Without "Noise" 68
b. With "Noise" 69
2. Rosenbrock's Function 73
a. Without "Noise" 73
b. With "Noise" 75

V POLICY COMPARISON (PC) 78

A. Problem Definition 78
B. Policy Comparison vs. Policy Iteration 79
C. Performance Based Policy Comparison - Monte Carlo Simulation 83

VI MULTI-RESOLUTION STATE-SPACE DISCRETIZATION WITH A GENETIC ALGORITHM FOR FUNCTION APPROXIMATION AND POLICY COMPARISON PERFORMANCE ANALYSIS (MGAP) 85

A. Full Algorithm Description 87
B. Summary of Least-Squares Policy Iteration 88
C. Comparison of MGAP vs. LSPI 90

VII BENCHMARK DYNAMIC SYSTEM EXAMPLE - INVERTED PENDULUM 93

CHAPTER Page

A. Inverted Pendulum Model 94
B. Inverted Pendulum Cast as a Reinforcement Learning Problem 95
C. Numerical Results 97
1. Case 1: Q-Learning 99
2. Case 2: Q-Learning with AAG 101
D. Summary 106

VIII SIMPLE RECONFIGURABLE SYSTEM EXAMPLE - MORPHING AIRFOIL 108

A. Airfoil Model 111
B. Airfoil Cast as a Reinforcement Learning Problem 113
C. Numerical Results 116
1. Case 1: Q-Learning 117
a. Developmental Stage 1: Initial Testing 117
b. Developmental Stage 2: Q-Learning Parameter Testing 128
c. Developmental Stage 3: Four Morphing Parameter Testing 137
2. Case 2: Q-Learning with AAG 143
a. Developmental Stage 1 144
b. Developmental Stage 2 146
c. Developmental Stage 3 148
3. Case 3: Q-Learning with AAG and PC 150
a. Dimensionality 151
b. Monte Carlo Simulation 151
c. Value Function and Policy Analysis 153
4. Case 4: Q-Learning with MGAP 153
a. Approximation 154
b. Monte Carlo Simulation 157
c. Value Function and Policy Analysis 158
D. Summary 161

IX RECONFIGURABLE SYSTEM EXAMPLE - MORPHING WING 163

A. Wing Model 164
B. Wing Cast as a Reinforcement Learning Problem 166
C. Numerical Results 167

CHAPTER Page

1. Case 1: Q-Learning with AAG and PC 169
a. Dimensionality 169
b. Monte Carlo Simulation 169
2. Case 2: Q-Learning with GA and PC 171
a. Approximation 171
b. Monte Carlo Simulation 172
D. Summary 174

X UNUSUAL APPLICATION EXAMPLE - THERMAL LOCATION FOR AUTONOMOUS SOARING 175

A. Thermal Model 175
B. Thermal Location Cast as a Reinforcement Learning Problem176
1. Single Thermal 179
2. Multiple Thermals 180
C. Numerical Results 180
1. Single Thermal in 2-Dimensions Numerical Results . . 181
a. Case 1: Q-Learning with PC 181
b. Case 2: Q-Learning with AAG and PC 184
c. Case 3: Q-Learning with MGAP 190
2. Multiple Thermals in 2-Dimensions Numerical Results 197
a. Case 1: Q-Learning with PC 198
b. Case 2: Q-Learning with AAG and PC 201
c. Case 3: Q-Learning with MGAP 206
3. Multiple Thermals in 3-Dimensions Numerical Results 215
a. Case 1: Q-Learning with AAG and PC 217
b. Case 2: Q-Learning with GA and PC 222
D. Summary 225

XI CONCLUSIONS 227

XII RECOMMENDATIONS 230

REFERENCES 233

VITA 262

LIST OF TABLES

TABLE		Page
I	GA for Function Approximation Parameters Settings	61
II	GA for Function Approximation Basis Function Sets	63
III	Knot Locations for Spline Interpolation Approximation of $y = 2x^2 + x - 3$	70
IV	Policy Comparison Example - Part 1	80
V	Policy Comparison Example - Part 2	81
VI	Constants of the Inverted Pendulum System	95
VII	Inverted Pendulum Q-Learning Example Reward Structure	96
VIII	Inverted Pendulum Q-Learning Parameters	97
IX	Inverted Pendulum Q-Learning Simulation Initial Conditions	98
X	Inverted Pendulum Policy Color Scheme	99
XI	Inverted Pendulum Action Sets and State Quantizations	101
XII	Inverted Pendulum AAG Goal Region Equations	103
XIII	Morphing Airfoil Axis Definitions	115
XIV	Morphing Airfoil Q-Learning Example Reward Structure	115
XV	Morphing Airfoil Parameter Limits	116
XVI	Reward Region for Morphing Airfoil Cases 1-3	118
XVII	Learning Parameter Constants for Morphing Airfoil Developmental Stage 1	118

TABLE Page

XVIII Distance Between Adjacent Vertices for Morphing Airfoil Developmental Stage 1 . . . 119

XIX Distance Between Adjacent Vertices for Morphing Airfoil Developmental Stage 2 . . . 129

XX Non-Goal States and State-Action Pairs for Morphing Airfoil Developmental Stage 2 . . . 131

XXI Distance Between Adjacent Vertices for Morphing Airfoil Developmental Stage 3 . . . 138

XXII Learning Goals for Morphing Airfoil Developmental Stage 3 . . . 138

XXIII Learning Parameter Constants for Morphing Airfoil Developmental Stage 3 . . . 139

XXIV Goal Series for Morphing Airfoil Developmental Stage 3 . . . 141

XXV Initial State of Agent for Examples 1-3 for Morphing Airfoil Developmental Stage 3 . . . 141

XXVI Airfoil AAG Parameters . . . 144

XXVII Morphing Airfoil Policy Color Scheme . . . 145

XXVIII States and State-Action Pairs for Morphing Airfoil AAG Developmental Stage 3 . . . 148

XXIX States and State-Action Pairs for Morphing Airfoil with AAG and PC 151

XXX Approximation Bit Strings for Each Level of Discretization for Morphing Airfoil . . . 155

XXXI Comparison of Data for Tabulated and Approximated Action-Value Function for Morphing Airfoil . . . 156

XXXII Morphing Wing Axis Definitions . . . 167

XXXIII Morphing Wing Parameter Limits . . . 167

XXXIV Wing Learning Parameters . . . 168

TABLE Page

XXXV States and State-Action Pairs for Morphing Wing Case 1 170

XXXVI Approximation Bit Strings for Only Level of Discretization for Morphing Wing 172

XXXVII Thermal Location Axis Definitions 177

XXXVIII Initial Updraft Field Limits 179

XXXIX Thermal Learning Parameters for Thermals in 2-Dimensions 182

XL Thermal Policy Color Scheme 184

XLI States and State-Action Pairs 187

XLII Approximation Bit Strings for Each Level of Discretization for Single Thermal in 2-Dimensions 192

XLIII Comparison of Data for Tabulated and Approximated Action-Value Function for Single Thermal in 2-Dimensions 193

XLIV Multiple Thermals in 2-Dimensions with AAG and PC: States and State-Action Pairs 203

XLV Multiple Thermals in 2-Dimensions with AAG and PC: Thermal Locations 205

XLVI Multiple Thermals in 2-Dimensions with AAG and PC: Number of Episodes Used 206

XLVII Approximation Bit Strings for Each Level of Discretization for Multiple Thermals in 2-Dimensions 209

XLVIII Comparison of Data for Tabulated and Approximated Action-Value Function for Multiple Thermals in 2-Dimensions 210

XLIX Multiple Thermals in 2-Dimensions with MGAP: Thermal Locations 211

L Multiple Thermals in 2-Dimensions with MGAP: Number of Episodes Used 212

LI Thermal Learning Parameters for Thermals in 3-Dimensions 217

TABLE Page

LII Multiple Thermals in 3-Dimensions with AAG and PC: States and State-Action Pairs 219

LIII Multiple Thermals in 3-Dimensions with AAG and PC: Thermal Locations 220

LIV Multiple Thermals in 3-Dimensions with AAG and PC: Number of Episodes Used 222

LV Approximation Bit Strings for Only Level of Discretization for Multiple Thermals in 3-Dimensions 224

LIST OF FIGURES

FIGURE		Page
1	1-Dimensional State-Space with Overlaying Pseudogrid	46
2	2-Dimensional State-Space with Overlaying Pseudogrid	47
3	4-Dimensional State-Space with Overlaying Pseudogrid	48
4	Multi-Resolution State-Space Discretization – Phase 1: Coarse Grid, Large Goal Range	50
5	Multi-Resolution State-Space Discretization – Phase 2: Finer Grid, Smaller Goal Range	51
6	Common Genetic Algorithm Operators	58
7	2^{nd} Degree Polynomial - (a) Fitness and (b) Average Distance	68
8	2^{nd} Degree Polynomial with Noise - (a) Fitness and (b) Average Distance	71
9	Comparison of the True and Approximated Data for $y = 2x^2 + x - 3$	72
10	Rosenbrock's Function - (a) Fitness and (b) Average Distance	73
11	Rosenbrock's Function with Noise - (a) Fitness and (b) Average Distance	75
12	Comparison of the True and Approximated Data for Rosenbrock's Function	77
13	Flowchart of Q-Learning	86
14	Flowchart of Proposed Algorithm	89
15	Flowchart of LSPI	91
16	Inverted Pendulum System	93

FIGURE Page

17 Basic Inverted Pendulum Controller: Value Function (a) and Policy Representation (b) 100

18 Basic Inverted Pendulum Controller, 10 Seconds: Time History (a) and Phase Diagram (b) 101

19 Basic Inverted Pendulum Controller, 300 Seconds: Time History (a) and Phase Diagram (b) 102

20 AAG Inverted Pendulum Controller: Quantized States (a) and Goal Regions (b) 103

21 AAG Inverted Pendulum Controller: Value Function (a) and Policy Representation (b) 104

22 AAG Inverted Pendulum Controller, 10 Seconds: Time History (a) and Phase Diagram (b) 105

23 AAG Inverted Pendulum Controller, 300 Seconds: Time History (a) and Phase Diagram (b) 106

24 Representative Fighter Aircraft - F-16 Fighting Falcon 108

25 Representative Bomber Aircraft - B-1B Lancer 109

26 Representative General Aviation Aircraft - Cessna 172 109

27 Representative Airfoil 110

28 $Q(s, a)$ Evolution for an Increase in Camber for Case #1 119

29 Monte Carlo Simulation for Q-learning of Case #1 120

30 Initial and Final Airfoil Configuration for Case #1 121

31 State Progression Using Greedy Policy for Case #1 122

32 $Q(s, a)$ Evolution for an Increase in Camber for Case #2 123

33 Monte Carlo Simulation Q-Learning of Case #2 124

34 Initial and Final Airfoil Configuration for Case #2 125

FIGURE Page

35 State Progression Using Greedy Policy for Case #2 126

36 $Q(s, a)$ Evolution for an Increase in Camber for Case #3 127

37 Monte Carlo Simulation for Q-Learning of Case #3 128

38 Initial and Final Airfoil Configuration for Case #3 129

39 State Progression Using Greedy Policy for Case #3 130

40 Policy Comparison for Each h_{x_i}: a) $h_{x_i} = 0.10\%$, b) $h_{x_i} = 0.25\%$, c) $h_{x_i} = 0.50\%$, and d) $h_{x_i} = 1.00\%$ 132

41 h_{x_i} Comparison: a) 100% Exploration, b) Anneal ε, c) Anneal ε and γ 133

42 Value Functions for 100% Exploration: a) $h_{x_i} = 0.10\%$, b) $h_{x_i} = 0.25\%$, c) $h_{x_i} = 0.50\%$, and d) $h_{x_i} = 1.00\%$ 135

43 Value Functions for Annealing of ε: a) $h_{x_i} = 0.10\%$, b) $h_{x_i} = 0.25\%$, c) $h_{x_i} = 0.50\%$, and d) $h_{x_i} = 1.00\%$ 136

44 Value Functions for Annealing of ε and γ: a) $h_{x_i} = 0.10\%$, b) $h_{x_i} = 0.25\%$, c) $h_{x_i} = 0.50\%$, and d) $h_{x_i} = 1.00\%$ 137

45 Agent Learning Success Results: a) $c_l \geq 0.4$, b) $c_l = 0.0 \pm 0.05$, c) $c_l = -0.2 \pm 0.05$, and d) $c_l = 0.2 \pm 0.05$ 140

46 Airfoil Lift Coefficient Results for Morphing Airfoil Developmental Stage 3 142

47 Airfoil Morphing Parameter Results for Morphing Airfoil Developmental Stage 3 143

48 AAG Airfoil Developmental Stage 1: Value Function (a) and Policy Representation (b) 145

49 AAG Airfoil Developmental Stage 2: Value Function (a) and Policy Representation (b) 147

50 AAG Airfoil Developmental Stage 3: Monte Carlo Simulation Results 149

FIGURE Page

51 AAG Airfoil Developmental Stage 3: Value Function (a) and Policy Representation (b) 150

52 Airfoil with AAG and PC: Monte Carlo Simulation Results 152

53 Airfoil with AAG and PC: Value Function (a) and Policy Representation (b) 154

54 Airfoil with MGAP: Monte Carlo Simulation Results 158

55 Airfoil with MGAP, Episode 2400: Approximate Value Function (a) and Policy Representation (b) 159

56 Airfoil with MGAP, Episode 3600: Approximate Value Function (a) and Policy Representation (b) 160

57 Airfoil with MGAP, Episode 8600: Approximate Value Function (a) and Policy Representation (b) 161

58 Airfoil with MGAP, Episode 8600: Value Function (a) and Policy Representation (b) 162

59 Representative Wing 163

60 Wing with AAG and PC: Monte Carlo Simulation Results 170

61 Wing with GA and PC: Monte Carlo Simulation Results 173

62 Representative Updraft Field 177

63 Updraft Field for Single Thermal in 2-Dimensions Case 1 182

64 Single Thermal in 2-Dimensions with PC: Monte Carlo Simulation Results 183

65 Single Thermal in 2-Dimensions with PC: Value Function (a) and Policy Representation (b) 185

66 Updraft Field for Single Thermal in 2-Dimensions Case 1 186

67 Single Thermal in 2-Dimensions with AAG and PC: Visited States in the Updraft Field 187

FIGURE Page

68 Single Thermal in 2-Dimensions with AAG and PC: Monte Carlo Simulation Results 188

69 Single Thermal in 2-Dimensions with AAG and PC: Value Function (a) and Policy Representation (b) 189

70 Updraft Field for Single Thermal in 2-Dimensions Case 3 190

71 Single Thermal in 2-Dimensions with MGAP: Visited State in the Updraft Field 191

72 Single Thermal in 2-Dimensions with MGAP: Monte Carlo Simulation Results 194

73 Single Thermal in 2-Dimensions with MGAP, Episode 600: Approximate Value Function (a) and Policy Representation (b) 195

74 Single Thermal in 2-Dimensions with MGAP, Episode 1200: Approximate Value Function (a) and Policy Representation (b) 196

75 Single Thermal in 2-Dimensions with MGAP, Episode 2400: Approximate Value Function (a) and Policy Representation (b) 197

76 Single Thermal in 2-Dimensions with MGAP, Episode 8600: Value Function (a) and Policy Representation (b) 198

77 Updraft Field for Multiple Thermals in 2-Dimensions Case 1 199

78 Multiple Thermals in 2-Dimensions with PC: Monte Carlo Simulation Results 200

79 Multiple Thermals in 2-Dimensions with PC: Value Function (a) and Policy Representation (b) 200

80 Updraft Field for Multiple Thermals in 2-Dimensions Case 2 201

81 Multiple Thermals in 2-Dimensions with AAG and PC: Visited State in the Updraft Field 202

82 Multiple Thermals in 2-Dimensions with AAG and PC: Monte Carlo Simulation Results 204

FIGURE Page

83 Multiple Thermals in 2-Dimensions with AAG and PC: Value Function (a) and Policy Representation (b) 207

84 Updraft Field for Multiple Thermals in 2-Dimensions Case 3 207

85 Multiple Thermals in 2-Dimensions with MGAP: Visited State in the Updraft Field 208

86 Multiple Thermals in 2-Dimensions with MGAP: Monte Carlo Simulation Results 211

87 Multiple Thermals in 2-Dimensions with MGAP, Episode 1400: Approximate Value Function (a) and Policy Representation (b) . . . 213

88 Multiple Thermals in 2-Dimensions with MGAP, Episode 2000: Approximate Value Function (a) and Policy Representation (b) . . . 214

89 Multiple Thermals in 2-Dimensions with MGAP, Episode 2400: Approximate Value Function (a) and Policy Representation (b) . . . 215

90 Multiple Thermals in 2-Dimensions with MGAP, Episode 8600: Value Function (a) and Policy Representation (b) 216

91 Updraft Field for Multiple Thermals in 3-Dimensions Case 1 218

92 Multiple Thermals in 3-Dimensions with AAG and PC: Visited State in the Updraft Field 219

93 Multiple Thermals in 3-Dimensions with AAG and PC: Monte Carlo Simulation Results 221

94 Updraft Field for Multiple Thermals in 3-Dimensions Case 2 223

95 Multiple Thermals in 3-Dimensions with GA and PC: Monte Carlo Simulation Results 225

CHAPTER I

INTRODUCTION

A. Introduction to the Class of Systems

A number of definitions can be applied to the term "highly reconfigurable systems". This term could refer to how a computer processes information or how a machine such as a copier processes a print job. It could also refer to a system of coordinated ground, air, and water vehicles that work together to achieve a mutual goal. *The all encompassing theme of these examples is that a highly reconfigurable system is one that can efficiently perform a variety of tasks that it could not achieve without reconfiguration.*

A particular subclass, and the class of particular interest in this research, is highly reconfigurable systems that are geometrically reconfigurable. This refers to a vehicle (air, ground, or water) or collection of vehicles that can change its geometrical features, i.e. shape or formation, to perform tasks that the vehicle could not otherwise accomplish. In aerospace terms a morphing aircraft or a formation of aircraft is an example of a highly reconfigurable system that can fly efficiently and well in any and all flight phases by changing shape parameters. At present there are only a relatively small number of physical realizations of this idea. This dissertation is addressed to establishing needed learning and control methodologies that are a fundamental enabler for reconfigurable systems.

Morphing aircraft have been of intense interest for engineers for many years. Aircraft are usually designed for optimal performance at only a few flight conditions. Thus, aircraft have particular purposes: fighter, bomber, general aviation, reconnais-

---

The journal model is *IEEE Transactions on Automatic Control.*

sance, *etc.* Optimizing over an entire range of flight conditions would allow a single aircraft to fill several of these rolls. A definition adopted here: morphing entails a $> 50\%$ change in geometry and parameters. Early examples of "morphing" or variable geometry aircraft include the Grumman F-14 Tomcat and the B-1 Lancer. Both have swing wings that are designed to have good performance at both cruise and supersonic flight when they are unswept and swept, respectively. Generally, previous physical realizations of morphing designs only considered changing one parameter to facilitate the aircraft flying in a particular condition.[1]

B. Literature Review

This research focuses on two main areas: reconfigurable air vehicles and discretization methods for reinforcement learning.

1. Reconfigurable Air Vehicle Literature Review

The focus has shifted in recent years to morphing aircraft with large-scale (i.e. 50% change in important geometric parameters) continuous shape changing rather than the purely mechanical or swing joints as seen before. Such reconfigurable systems provide a wealth of issues for researchers to investigate. Many of the broad areas of concern are overviewed in References [1, 2, 3]. Reference [1] qualitatively describes how each individual wing geometric parameter affects aircraft performance and presents challenges in each aspect of morphing: optimal design configurations, structures, actuation, aerodynamics, controls, propulsion, and the integration of subsystems. Jha *et al.* conclude that for a true multi-mission morphing aircraft, several wing parameters need to be changed simultaneously. Reference [2] takes a slightly different perspective and describes the need to outline mission requirements and ve-

hicle capability assessments. Then morphing is brought in to determine how to meet these requirements. A more pragmatic approach is to consider an affordability comparison between morphing and non-morphing aircraft.[3] Such a comparison shows that the adaptive and multifunctional technologies a morphing aircraft would employ has better affordability.

Developing mission requirements and effectiveness measures for this class of system takes on new facets as compared to fixed geometry vehicles. The flexibility and variability of the these systems means in effect that many different types of missions can be accomplished by one type of vehicle. Identifying what kinds of missions would mesh well and how to coordinate them is of interest. A scenario based evaluation model can be used to compare a fleet of morphing aircraft with a fleet that has many different aircraft to determine optimal missions for such systems.[4, 5] Other design methods have been used to develop missions for this class of systems, such as the affinity method, brainstorming, weight objectives, and quality function deployment, and has been shown to be effective.[6] Evaluating such missions is of interest to the authors of References [7, 8, 9]. Methods ranging from probability measures of mission success[7], creating "surrogate models" to compare efficiency, cost, and capability of a single aircraft or a fleet[8], or just direct comparison between morphing and non-morphing vehicles for a particular mission[9].

a. Design of Highly Reconfigurable Systems

Designing a highly reconfigurable system to accomplish many disparate mission requirements becomes a difficult task in that no longer is the designer attempting to optimize one phase of flight and perform only adequately during other phases. The geometric parameters that the designer wishes to change during operation must be determined, and the vehicle must essentially be redesigned for every flight phase that

must be optimized. This problem becomes a multiobjective optimization problem with the added challenge of introducing reconfigurable shape parameters. Several different methods have been developed to address these design challenges, a few of which are described in References [10, 11, 12, 13]. Of course, it is one thing to have optimal geometry for a range of flight conditions and another to have an aircraft design capable of achieving the desired geometry changes. Thus the ultimate design challenge is much more complicated. We must invent mechanisms of reconfiguration, including actuation, that can achieve the geometry changes and satisfy other requirements on bearing associated loads, maintain stability, *etc.* Obviously, achieving geometric reconfigurability almost always comes with a mass penalty. It is important to recognize the united role that mechanism design plays in geometrically morphing systems.

Integral to the design of this class of systems is sizing. Morphing is treated as an independent variable that must be optimized during aircraft sizing in References [14, 15]. These references cast the performance, size, and weight of the aircraft as functions of morphing and attempt to answer the question of what geometric features should be changed and by how much. One study developed a sizing approach through the use of continuous optimization (i.e. using calculus-of-variations-based optimization methods).[16] To be physically meaningful, conceptual sizing optimization should consider the dependence of size and weight of the wing on the range of reconfigurability. References [17, 18] develop wing weight equations to aid in aircraft sizing by using response surface methods and emphasizing the actuating mechanism, respectively. The authors use finite element-based optimization methods for morphing wings to predict wing weight.

There are many recent examples of vehicles in this class of systems and how they will effect the aircraft industry. Described are only a handful that should give an idea

of how far reaching these can be. Hong *et al.* develop an unmanned combat aerial vehicle that use shape changing to meet the requirements of a low-speed, high-altitude loiter and a supersonic strike capable aircraft.[19] The designed aircraft changes wing area by 200%, sweep angle from 20° to 70°, and aspect ratio from 3 to 7. The Daedalon, a design baselined for Mars, transforms itself from a blunt-body entry spacecraft into an airplane through a form of wing morphing to allow for a flexible architecture for unmanned planetary exploration.[20] References [21, 22] consider the impact of incorporating this class of systems into current commercial aircraft service.

b. Wing Reconfiguration

References [23, 24, 25, 26, 27, 28] focus specifically on the design and optimization of morphing wings, the component to most likely be changed in aircraft that are highly reconfigurable. Nangia and Palmer develop a wing design method that can design throughout the flight envelope and incorporate such parameters as wing sweep, area, and aspect ratio.[23] Range and endurance performance is also addressed. Vale *et al.* use simple aerodynamics and a sequential quadratic programming optimization algorithm to optimize the aerodynamics of a morphing wing.[24] References [25, 26] describe the design, development, and testing of a pneumatic telescopic wing to change aspect ratio. The wing is tested for lift, drag, and lift-to-drag ratio at various configurations. Cadogan *et al.* describe the design of an inflatable wing and actuation schemes to aid in developing compact package unmanned aerial vehicles (UAVs).

In addition to the time and parameter varying forces and inertias, reconfiguring the wing presents the added complication of changing aeroelastic characteristics. Reference [29] investigates the migration of the wing flutter boundary for a cantilevered wing undergoing changes in span. Bae *et al.* also consider a variable span wing.[30] They determine that deformation due to bending is much more significant than defor-

mation due to twist and recommend that bending stiffness in morphable wings must be carefully considered. Reference [31] seeks to develop a structural and aeroelastic model suitable for investigating actuation concepts for a reconfigurable vehicle, and Reference [32] investigates the applicability of existing aeroelastic simulation techniques in the analysis of morphing aircraft design.

c. Structure of Reconfigurable Systems

The underlying structure of this class of systems is of particular interest. The question of how one is to achieve reconfiguration often arises. Love *et al.* address this broad concept in Reference [33] by highlighting that new actuator systems must be designed to achieve reconfiguration and handle aircraft flight speed, maneuver load factor, and actuator response. These provide sensitivities in structural weight, aeroelastic load factor, and actuator flight load distributions. Reference [34] takes a similar broad approach and seeks to develop an evaluation matrix that highlights structural morphing concepts. The authors discuss several concepts, such as sliding plates, articulated sections, *etc.*

Several previous studies have considered specific structures and mechanisms designed to achieve reconfiguration. Reference [35] describes the mechanism to morph a hyper-elliptic camber span wing. The mechanism is essentially a scissor linkage in which an input is applied to the first segment and the rest of the segments move in reaction. Reference [36] uses a series of cables, rather than a scissor linkage, to form a tendon-actuated compliant truss to achieve desired shapes. Obviously, in addition to enabling the range of geometric variability, this mechanism and the overall structure must remain capable of withstanding structural stability over the entirety of the configuration space. Another method seeks to minimize energy input by creating an airfoil-like active bistable twisting structure that changes shapes by a snap-through

action.[37] Reference [38] focuses on means to morph, and specifically considers the materials that should be used. The authors investigate such materials as shape memory alloys and polymers, piezoelectric materials, and electro- and magneto-rheological materials. References [39, 40] created a genetic algorithm that uses load path representation to design the optimal internal structure of a reconfigurable vehicle. The algorithm seeks to minimize the shape deviation between the deformed shape and the desired target shape.

References [41, 42, 43] consider the issues of the effects of stiffness on control power, the minimization of actuator and structural loads, and actuation power requirements for morphing wings, respectively. Wing structural stiffness obviously affects the energy required to deform the wing. Care must be taken to account for avoiding flutter and still retaining the ability to deform the structure.[41] Changing from one configuration to another is a delicate matter as the forces on the surface change in response. Minimizing these forces and the subsequent loads on the actuators and internal structure is the focus of Reference [42].

Two successful applications of shape memory alloys used to achieve structural morphing are Boeing's variable geometry chevron and a rotor blade twist control system by Bushnell *et al.* The chevron change shape at takeoff, approach, and cruise to meet acoustic test requirements.[44] The rotor blades change twist to achieve better lift characteristics during flight.[45]

d. Performance, Simulation, Control, and Testing of Highly Reconfigurable Systems

Introducing reconfiguration can greatly impact the performance of the vehicle in question. Understanding and evaluating this impact can lead to better vehicle and mission designs. Bowman *et al.* conduct a mathematical evaluation of bird morphing mechanisms. They consider how variable lift-to-drag ratios and specific fuel consumption

can affect performance on maneuvers such as turn radius. Reference [46] describes how performance can be minimized or maximized by varying the planform variables of a wing. Reference [47] poses the design of a wing as a multilevel, multiobjective optimization problem with the competing performance objectives of maneuverability and long range/endurance.

Simulation of this class of vehicles also presents new challenges. Changing the geometry of a vehicle introduces terms not seen in rigid body equations of motions. Additional terms may include parameter and/or time varying aerodynamic forces and inertias. References [48, 49, 50, 51, 52, 53] describes various research investigating simulation techniques for reconfigurable vehicle dynamics. Each tool considers changing such parameters as wing sweep, span, area, *etc.*, and their effect on the dynamics of the vehicle. Some even consider the challenge of using a combination of morphing parameters for pitch, roll, and yaw control.[51, 52]

Controlling this class of vehicles on the macro scale was investigated in References [54, 55, 56, 57]. Reich and Bowman mention using dynamic inversion and classical proportional-integral-derivative (PID) controls in their simulations.[48, 50] Baldelli *et al.* describe the modeling and control of an aeroelastic morphing vehicle in which the controller uses a set of inner-loop gains, designed via classical techniques, to provide stability and a linear parameter-varying outer-loop controller for robust stability and performance for the time-varying dynamics. Whitmer and Kelkar design an H_∞ controller to achieve robustness with respect to multiplicative and parametric uncertainty for a morphing wing, commanded to track lift and roll moments.[55] Ataei-Esfahani and Wang design a probabilistic robust explicit-model-following controller to assure the stability of a morphing aircraft model subject to uncertain actuator failure.[56] Tao *et al.* also consider the case of uncertain actuator failures in morphing aircraft. Their approach uses an adaptive compensation scheme to achieve desired closed-loop

stability and tracking.

Implicit in the approach of many researching this class of vehicles is using morphing to achieve effective actuation that accomplishes what control effectors do in rigid body vehicles. For example, Sanders *et al.* compare the roll performance between a conventionally controlled wing and a wing with morphing or conformal control surfaces.[58] By using twist and deforming trailing-edge control surfaces with the aid of smart materials, conformal surfaces can be used for aerodynamic control.[58] Reference [59] investigates the energy required to use morphing for aerodynamic control as compared to conventionally controlled wings. The authors' results indicate that morphing aircraft can outperform conventional aircraft in terms of required flight control energy.[59]

The next step in the development of any new vehicle technology is to test the vehicle in a controlled environment. In the case of aircraft, this means wind tunnel testing of part or all of the designed morphing aircraft. References [60, 61, 62, 63] describe the development of their respective wind tunnel models and the results of the testing. Reference [60] continues the development of the pneumatic telescopic spar concept of Blondeau *et al.* Guiler *et al.* seek to show that morphing can be used to improve the flight characteristics of an aircraft with unusual configuration; a tailless aircraft in this case.[62] Of particular interest is the work by Boria *et al.* The authors set up a hardware-in-the-loop optimization scheme that uses a genetic algorithm to optimize the airfoil of the wing as it sweeps through angle-of-attack.[63]

e. Morphing Airfoil

The problem of a morphing airfoil in particular was investigated by Hubbard in Reference [64]. Hubbard focuses on the physical shape change of an airfoil modeled by a space/time transform parameterization. The space/time parameterization results in

a spatially decoupled system with Fourier coefficients as inputs and orthogonal basis shapes as outputs.[64] Reference [65], however, use postbuckled spinal structures for manipulating the shape of the airfoil with the goal of achieving certain aerodynamic performances with only a minimal amount applied load needed to change airfoil shape. References [66, 67, 68, 69] all seek to optimize a reconfigurable airfoil by either optimizing the static aerodynamic shape for various flight phases or by optimizing the shape change itself with respect to actuation energy and/or aerodynamic drag.

f. Biologically Inspired Morphing Aircraft

Biologically inspired morphing is of great interest in the realm of a particular subclass of this class of systems: micro air vehicles (MAVs). Due to their small size and membrane lifting surfaces, they often do not have conventional control surfaces, especially on the wing. Thus other means of control must be investigated. Roll control is of particular interest. In lieu of ailerons, torque rods attached to the membrane of the MAV wing as well as possible other aeroservoelastics controls cause the wing to either twist, curl, or both. Flight tests show that wing twist and/or curl provide an excellent strategy to command roll maneuvers.[70, 71] The torque rods used to achieve wing twist are further optimized in Reference [72] using genetic algorithms in which a vortex lattice method is used to determine fitness. The flight dynamics of the test vehicle show that turns and spins can also be repeatedly performed and that significant control authority is also provided for lateral dynamics.[71, 73] An additional degree of morphing in the form of wing sweep is added to the wing dihedral morphing parameter in Reference [74]. This extra degree is shown to have considerable effect on the handling qualities and stability of the vehicle.[74] Morphing just the dihedral angles is also shown to have an effect on flight performance metrics such as climb rate, glide angle, and stall characteristics.[75] The effect of dihedral morphing on per-

formance metrics and dynamics is further investigated using a vortex lattice method to computationally determine the aerodynamics of the MAV.[76] The optimal wing geometries are found to converge to biological solutions in several instances.[76] The dynamics are also examined further using a vortex lattice method for the case of wing sweep morphing. The MAV is shown to have enhanced turning capabilities with this morphing parameter as well as enhanced crosswind rejection.[77]

Other concerns for biologically inspired morphing in micro air vehicles are the structural effects of deformation and the control of the vehicle. The structural effects for the vehicle discussed in the previous paragraph are outlined in Reference [78]. The authors examine the relationship between wing flexibility and performance and demonstrate the wing under loading in a wind tunnel in association with lift and drag characteristics.[78] Controllers for MAVs, on the other hand, are discussed in References [79] and [80]. Reference [79] considers the inherent nonlinearity in the closed-loop equations of motion of a morphing vehicle. Focusing on disturbance rejection, the authors develop a proportional feedback controller and a nonlinear Lyapunov controller and apply them to several types of morphing.[79] A controller for the MAV discussed above is developed in Reference [80]. The authors consider the case of variable wing dihedral in which they use a vortex lattice method to calculate the aerodynamics.[80] Desired dynamics for each mission phase are chosen and H_∞ model-following controllers are developed for each.[80] Simulations show favorable results in maintaining the control and stability of the vehicle.[80]

2. Reinforcement Learning Literature Review

The majority of the references listed above only address two of the three essential functionalities of this class of systems: when to reconfigure and how to reconfigure. Learning to reconfigure, however, is only occasionally discussed. Learning to reconfig-

ure entails the actual physical change from one shape to another. Optimization and cost functions have been used to determine how to accomplish reconfiguration, but these actuation schemes are still in the early stages of understanding. A candidate approach for learning to reconfigure is just that...*learning*, more specifically to use some algorithm for *machine learning*. For this class of systems, reinforcement learning in particular can be used to learn both good combinations of the geometric parameters *and* to change from one set to another. Reinforcement learning is a form of machine learning that gathers information and learns by interacting with an environment.[81] This eliminates the need to find a single or a handful of vehicle configurations and designing control laws to change from one to another.

Reinforcement learning and related techniques have been topics of interest as well. New algorithms, additions to existing algorithms, and modifications to algorithms are constantly being developed. To understand the contribution of the algorithm developed in this dissertation, I will first overview the current state of knowledge.

a. Dynamic Programming

Closely related to reinforcement learning is dynamic programming, which in this context is a method of solving problems with an overlapping structure or as a set of subproblems. This approach involves a first step of splitting the problem into the simplest possible subproblems, and then solving them step by step. The central equation for this method is the dynamic programming equation or Bellman equation. An incarnation of this equation is shown below in Eq. 1.1

$$\max_{\{x_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t F\left(x_t, x_{t+1}\right) = V\left(x_0\right) \tag{1.1}$$

where V is the optimal value that can be obtained by maximizing this function subject to the constraints, x_0 is the initial situation, x_t is the state at time t, x_{t+1} is the state

at time $t + 1$, and β is the discount factor. This method of solving optimization problems writes the value of a decision problem at a given point in time in terms of a payoff from the initial situation and constraints and the value of the remaining decision problem that results.

Bertsekas has done much work with dynamic programming over the years. Bertsekas has integrated dynamic programming with suboptimal control in the form of rollout and model predictive control.[82] Distributed dynamic programming is an offshoot in which several processors participate in solving dynamic programming problems that Bertsekas has investigated.[83] Also, he developed separable dynamic programming that solves problems in separate subsystems that are tied together by a common constraint.[84] Differential dynamic programming and its discrete counterpart, on the other hand, is used to determine the optimal control problem function of a nonlinear system.[85, 86] These methods have been applied to such problems as bang-bang control[85] and optimal orbit transfer[86].

Neuro-dynamic programming is yet another class of dynamic programming, but this method is meant for control and sequential decision making under uncertainty.[87] Bertsekas and Tsitsiklis describe the method in Reference [87] as a method that "combines ideas from the fields of neural networks, artificial intelligence, cognitive science, and approximation theory." Similarly, References [88, 89] use approximate dynamic programming to design neural network-based feedback controllers. Reference [90] describes a method of shifting the approximate dynamic programming control problem to a higher level such that the algorithm does not design controllers but rather selects design from a cache of existing controllers.

Other areas in dynamic programming include developing value iteration and Q-learning methods for the average cost dynamic programming problem.[91] Atkeson *et al.* seek to find steady-state policies for control problems in Reference [92].

The authors consider various research topics on approximate dynamic programming. Reference [93] describes a particular application of direct heuristic dynamic programming: a large power system stability control problem. This method is used to address nonlinear coordinated control under uncertainty.

Another form of learning is locally weighted learning. This type of learning remembers experiences explicitly. Predictions and generalizations are then performed in real time by building a local model to answer any particular query. Reference [94] describes how the authors use this form of learning on control tasks.

b. Iterative Learning Control

Iterative learning control (ILC) is an iterative approach to tracking control for repetitive processes. Reference [95] reports a complete analysis of the learning control problem for linear, time-invariant (LTI) plants and controllers. The authors apply ILC to several examples. Hatonen *et al.* consider the discrete-time case from the algebraic point of view.[96] They also present a general convergence theory of ILC systems. Reference [97] also considers the discrete-time case. The authors present a stability analysis of ILC problems with interval uncertainty. Chien in Reference [98] considers the challenge of applying ILC to nonlinear systems. This reference introduces a new adaptive fuzzy iterative learning controller to deal with disparate control tasks of nonlinear systems, state errors, and input disturbances. The fuzzy component acts as an approximator to compensate for plant nonlinearity, while the robust learning component acts on the input gain, input disturbance, and fuzzy approximation error. This method is a good example of useful additions and modifications to an existing learning algorithm.

c. Fuzzy Q-Learning

Another example of modifying or adding to a pre-existing learning algorithm is fuzzy Q-learning. Q-learning is a reinforcement learning method that forms the core of this dissertation and will be explained in more detail in subsequent sections. Fuzzy Q-learning can be thought of in two ways. First, the goal can be to find the best set of fuzzy rules and change the weights associate with those rules. This method essentially develops a fuzzy inference system.[99, 100] The second method is to represent the learning parameters associated with Q-learning and/or the states themselves as fuzzy rules.[101, 102] This method better approximates continuous state-spaces as compared to traditional Q-learning.

There are several successful applications of fuzzy Q-learning. Reference [103] describes an agent learning to play soccer using fuzzy Q-learning. The authors show that the agent learns good offensive behavior in the game. References [104, 105] use an adaptation of fuzzy Q-learning aid performance when large disturbances and environment variations are present. The authors apply this method to robot manipulators and state that this controller outperformed other controllers.

d. Methods Related to Q-Learning

There are many related reinforcement learning topics and areas of research that do not lend themselves to easy categorization. These references will be discussed in the following several paragraphs. The first is Reference [106], which addresses the problem of learning a policy in simulation that does not work in real-life. This phenomenon is often due to inaccurate models of complex tasks on which the agent learns. This reference presents a reinforcement learning algorithm that only needs an approximate model that is then locally modified using real-life trials. Empirical results show good

performance in the real system.

Reference [107] describes a method to stabilize or regulate a system using Lyapunov design methods. The agent learns to control the system by switching between basic controllers and learning what does and does not work for the given situation. This reference is a good example of how reinforcement learning can be successfully applied to higher level problems rather than learning the low level controller itself.

Tan *et al.* constructed a new method that learns a value function similar to reinforcement learning, but it estimates it using a combined architecture of several different methods, such as Sarsa and Q-learning.[108] Reference [109] also develops such an ensemble method. The author seek to enhance performance by combining action selection (the policy) of several different reinforcement learning algorithms. The algorithms used are Q-learning, Sarsa, actor-critic, WV-learning, and actor-critic learning automation. The policies are derived using several different ensemble methods. Results show the ensemble methods outperform the single reinforcement learning algorithms.

References [110, 111] are concerned with quantum Q-learning and quantum reinforcement learning, respectively. These methods use principles of quantum computing to aid the learning process. Probabilistic methods are used in this approach to the learning process. The states, actions, and success all have associated probabilities. Results show that these methods perform well while maintaining a good balance of exploration vs. exploitation (which will be discussed in more detail in later chapters).

e. Multiagent and Hierarchical Reinforcement Learning

Multiagent reinforcement learning is another topic of intense interest, and actually of particular importance to this class of systems as eventually there will be a call for coordinating a fleet of intelligent agents. Reference [112] surveys the state of the

art in multiagent reinforcement learning. The authors outline concerns such as how to formally define the goal, stability of agents' learning dynamics, and adaptation to behavior of other agents as they too learn. Reference [113] addresses some of these issues by developing new algorithms that learn a parameterized representation of a policy or value function. The agents can use this representation to determine "jointly optimal actions" without the need to explicitly consider every possible action. In contrast, Reference [114] presents a method in which one agent predicts the probability of the other agents' actions under a certain state and combines this with its own Q-value (the agents learn via Q-learning) to choose its action policy.

Hierarchical reinforcement learning (HRL) is somewhat similar to dynamic programming in that HRL seeks to split a complex learning problem into a hierarchy of subtasks or subgoals that can be learned individually. Determining those subgoals can prove to be a challenge. Reference [115] discusses an algorithm that combines Q-learning and a locally weighted learning method to select behavioral primitives and generate subgoals for the agent. Reference [116] identifies subgoals by partitioning local state transition graphs. Reference [117] develops the theory of quad-Q-learning. This method follows the "divide and conquer" mentality. Not strictly HRL, but the general thought behind the method is similar. The algorithm treats state transitions slightly differently than in traditional Q-learning. Rather than being in a state, choosing an action, and transitioning to some new state, when an action is chosen in quad-Q-learning either there is a reward and no transition or there is no reward and four new states result. Each new state is treated as a new environment, and learning commences in a similar fashion. Hierarchical partially observable Markov Decision Processes (MDPs) are described by Theocharous and Mahadevan.[118]

MAXQ is a popular HRL method that decomposes a learning problem into a hierarchy of subtasks to be learned using Q-learning. Thus it is a hierarchical Q-learning

algorithm. The method is first developed by Dietterich in Reference [119]. This paper defines the algorithm, proves convergence, and empirically shows that it learns faster the Q-learning alone. This algorithm is further developed into a model-free learning algorithm MAXQ-Q, which is also shown to learn faster than Q-learning.[120] Mehta *et al.* develop an algorithm that works in conjunction with MAXQ. This algorithm discovers MAXQ task hierarchies and subtasks by analyzing the relationships among actions. References [121, 122, 123] also contain MAXQ in their frameworks. Reference [122] integrates MAXQ with genetic programming, which explores possible hierarchies. Multiagent reinforcement learning and HRL are combined by Makar *et al.* with promising results.[123]

There are, of course, many other forms of HRL. Kirchner applies a hierarchical form of Q-learning to a six-legged walking machine.[124] Movement is split into the elementary swing and stance movements of individual legs and the overall coordination scheme to achieve forward movement. The highest layer uses these learned movements to achieve goals in the environment. Even the elementary swing and stance movement of the leg is split into four separate parts to reduce problem complexity. Reference [125] uses hierarchical Q-learning for local robot navigation. Reference [126] describes a different application of HRL in which Ghavamzadeh and Mahadevan develop a multiagent HRL algorithm that learns how to optimize necessary communication between agents such that they coordinate movements to achieve goals. Reference [127] uses HRL to represent behavior for efficient creative searches to mimic the evolution of human creative behavior in intelligent systems.

f. Learning on a Continuous Domain

A common factor to many forms of reinforcement learning, whether base algorithm, multiagent, or hierarchical, is that the state- or state-action space must often be

discretized into a finite number of states or state-action pairs. Discretizing a continuous state problem can lead to its own set of issues, and as a result there is a good amount of research focusing on developing methods to learn on a continuous domain. Often some sort of function approximation occurs during learning. Baddeley addresses one of the issues that arises with using function approximation during learning. The author states that often function approximation can disrupt what has been learned previously through negative interference.[128] He proposes learning the value function using a multilayer perceptron network, which is shown to improve the speed of learning. Other authors apply such solutions as piecewise linear function approximation[129], linear function approximation of factored MDPs[130], and advantage updating[131], which is use for reinforcement learning in continuous *time*.

Continuous learning methods have also been developed specifically for Q-learning as well. Reference [132] develops a region-based reward assignment with convex clustering of the region for Q-learning. The authors show that this allows robots to move smoothly from any arbitrary state. Reference [133] approximates the action-value function of Q-learning using regularization theory and radial basis functions to create continuous valued states and actions. Szepesvari and Smart use the total expected discounted cost and local function approximation to learn in continuous state spaces. Examples of continuous state Q-learning applied to robot systems include learning a biped gait using a spline based estimation algorithm with Q-learning[134], the region-based reward structure for robot navigation[135], and calculated contribution values to estimate a continuous action for a vision-guided robot[136].

A more general form of continuous learning method first emerged less than 15 years ago. This type of method bases a learning algorithm on the theory of linear least-squares function approximation. Bradtke and Barto introduced two new temporal difference (TD) algorithms based on this theory. These two are Least-Squares

TD (LSTD) and Recursive Least-Squares TD (RLSTD).[137] Nedic extends upon LSTD by showing convergence of LSTD(λ) with probability 1.[138] Bertsekas further explores the LSTD algorithm and similar algorithms in Reference [139] Lagoudakis and Parr take these methods a step forward by developing least-squares method for Q-learning.[140] This method is called LSTD Q-learning (LSTDQ of LSQ). This uses least-squares theory to approximate the action-value function $Q(s,a)$, rather than just the value function $V(s)$, which allows for action selection without a model. They also add policy iteration to LSTDQ to incrementally improve the policy. This method is called Least-Squares Policy Iteration (LSPI). The authors successfully apply this algorithm to an inverted pendulum, balancing and riding a bicycle, two-player zero-sum Markov games, and Tetris.[140, 141]

g. Applications of Machine Learning Algorithms

Q-learning and algorithms that solve reinforcement learning problems in general have been applied to numerous problems and systems. Reddy uses Q-learning to detect the presence of primary signals and for spectrum utilization.[142] Reference [143] describes the use of Q-learning for traffic signal control to minimize delays. Various reinforcement learning methods are also used to learn biped walking[144], robot juggling[145], air hockey and a marble maze[146], control of power systems[147], and engine torque and air-fuel ratio control[148], *etc.*

Machine learning has also been applied to several aerospace applications, though not strictly a part of the class of systems discussed above. Reference [149] applies reinforcement learning to the problem of control of an autonomous soaring aircraft. Kampen *et al.* have success using forms of Adaptive Critic Designs to control a model of the Lockheed Martin F-16 Fighting Falcon with changing plant dynamics.[150] Reference [151] describes the use of reinforcement learning to control Micro-Trailing Edge

Effectors to suppress flutter. References [152, 153] apply forms of reinforcement learning to autonomous helicopter control with impressive success while Reference [154] applies neural dynamic programming to a similar problem. Bertsekas *et al.* apply Neuro-dynamic programming to missile defense and interceptor allocation. Rather than control, Goel and Hajela use reinforcement learning for the complex aerodynamic optimization problem of turbine airfoil design.[155] One application that could nominally be considered a member of the class of systems in question is a flapping MAV. Reinforcement learning is used for lift generation and shown to converge to a flapping motion both in simulation[156] and in experiment[157].

h. Learning Applied to Morphing

The melding of morphing and learning, i.e. using machine learning to learn how to change the shape of a reconfigurable vehicle, has been investigated several times in the past in the form of Adaptive-Reinforcement Learning Control (A-RLC). Reference [158] describes a methodology that combines Structured Adaptive Model Inversion (SAMI) with Reinforcement Learning to address the optimal shape change of an entire vehicle. The method learns the commands for two independent morphing parameters that produce the optimal shape. The authors show that the methodology is capable of learning the required shape and changing into it and accurately tracking some reference trajectory.[158] This methodology is further developed in Reference [159]. It is extended to an "air vehicle" using Q-learning to learn the optimal shape change policy. The authors show that the methodology is able to handle a hypothetical 3-D smart aircraft that has two independent morphing parameters, tracking a specified trajectory, and autonomously morphing over a set of shapes corresponding to flight conditions along the trajectory.[159] Finally, the methodology is further improved upon by applying Sequential Function Approximation to generalize the learning from

previously experienced quantized states and actions to the continuous state-action space.[160] The authors showed that the approximation scheme resulted in marked improvements in the learning as opposed to the previously employed K-Nearest Neighbor approach. All of these examples, however, only have two independent degrees-of-freedom that must be learned. Even a small "real world" morphing problem will have several interdependent degrees-of-freedom. Learning to manipulate more morphing parameters creates a more complex learning problem.

C. Research Objectives/Motivation

Currently, reconfigurable systems are designed to perform well at a handful of operating conditions. For reconfigurable aircraft, these operating conditions are flight phases or specific flight conditions. Controllers are then designed to change the shape from one configuration to another while avoiding constraints such as geometric limitations, extreme structural loading, and vehicle stability. These controllers can be designed using optimal control techniques, classical control techniques, modern control techniques, *etc*, and are often sensitive to initial and final conditions. Due to noise in the system and other problems that could occur, it is likely that the configuration will not always change from one specific design point to another. The designed controllers could still successfully guide the reconfiguration, but to ensure a successful change, multiple controllers may be necessary to cover all of the possible configurations of the vehicle. An alternative is to use machine learning. Not only could machine learning, and more specifically reinforcement learning, learn the appropriate configurations for a wide range of operating conditions, it could also learn the policy to control the vehicle change from *any* configuration to the target configuration while taking into account the aforementioned constraints.

Therefore, this dissertation focuses on the learning of deterministic, highly non-linear, complex, high dimensional reconfigurable systems that can be cast as reinforcement learning problems. It seeks to investigate and understand the problem of efficiently learning complex state-spaces with particular attention to "Regions Of Interest" containing high concentrations of information, and the problem of efficiently storing and using the learned information.

The objectives of this research are the following:

1. Investigate methods to reduce the total number of states the agent must visit while maintaining the uniqueness of the goal.

2. Explores methods that maintain a high rate of convergence of the action-value function while maintaining the uniqueness of the goal. Reducing the number of states reduces the computation time as well as keeps the rate of convergence high, which is shown in simulation. This ability becomes important should the method be applied to more time intensive models.

3. Develop ways to aid in the approximation of the action-value function during learning to track the evolution of both the tabular action-value function and its approximation such that learning is terminated only when both the action-value function and its approximation have converged and a good policy learned.

4. Autonomously adjudicate between wide area exploration and Region Of Interest exploitation. The algorithm should recognize Regions Of Interest, learn that region, recognize when adequate learning has taken place, and recognize the need when learning in that region must continue.

D. Scope

The scope of this research investigates some unusual problems, such as the morphing airfoil, morphing wing, and thermal location for autonomous soaring as reinforcement learning problems. The morphing airfoil is investigated in particular detail such that the effects of various learning parameters on the learning itself is analyzed. Learning focuses on Regions Of Interest of the state-space containing higher concentrations of relevant information than elsewhere. This focus is extensible to a state-space with *multiple* Regions Of Interest. Approximation of the learned action-value function is incorporated into the learning algorithm. By periodically computing the function approximation and monitoring the convergence of the policy represented by the approximated action-value function, not only can an approximation that preserves the policy be found, so too can the number of learning episodes needed for convergence be kept to a minimum.

E. Contribution to Body of Knowledge

This research is novel and represents a fundamental advance for several reasons. Only recently has the shape changing aspect of the reconfigurable aircraft problem been posed as a reinforcement learning problem. Often several configurations of a reconfigurable system is designed, and then some control technique that determines how the system or vehicle changes from one configuration to another. Casting the reconfiguration as a reinforcement learning problem melds these two problems into one. By carefully defining the state-space, action space, constraints, and reward function, both the configurations for desired operating conditions and the control to change from one to another is learned. This dissertation presents several reconfigurable system applications of the learning techniques developed here that achieves such learned

reconfiguration.

A multi-resolution discretization method is a novel approach to improving Q-learning by gathering the most detailed information in and around Regions Of Interest, namely the goal. Q-learning on a continuous domain quickly becomes intractable when one considers that convergence of the algorithm to the optimal action-value function is only guaranteed if the agent visits every possible state an infinite number of times.[161] An agent would therefore visit an infinite number of states using an infinite number of actions an infinite number of times. Add in the fact that the states can be defined by anywhere from 1 to N continuous variables and the dimensionality of the problem becomes a significant issue. This multi-resolution method provides a means of learning the action-value function, $Q^{\pi}(s, a)$, for a fixed policy, π, in progressively finer detail. It seeks a compromise between the high rate of convergence of a coarse discretization, with the high level of detail of a fine discretization. It is desired that the method be extensible from 1- to N-Dimensions and be easily extended to other reinforcement learning problems similar to those presented in this dissertation.

Generally, the purpose of the multi-resolution methods in machine learning is to refine the value function, $V(s)$, where needed, such as when the agent finds itself in a difficult situation and the current resolution of the value function does not offer an escape. Finer resolutions could yield a path out of the situation. This is not the case with the method presented in this dissertation. The multi-resolution state-space discretization method is driven instead by identified Regions Of Interest. Once identified the method refines its search criteria and the state-space discretization iteratively until the agent can find a very specific goal and the path to it. Thus, enhancing Q-Learning with the multi-resolution technique developed in this dissertation is a powerful addition to the Q-Learning algorithm.

Conceptually, this method can be described by a simple example. Consider a

scout that is sent out initially to locate any enemy encampments. When the scout encounters an encampment, the scout maps out the perimeter and determines the best approach and retreat. The scout then has orders to penetrate the encampment, find the area housing the officers, and map the route to that area. The final refinement of the scout's orders is to locate and map a route to a particular tent. The scout has a series of progressively more specific orders that are carried out sequentially in a manner similar to this method.

This dissertation also develops a novel function approximation tool to aid in the approximation of the Q-Learning action-value function. This tool is a genetic algorithm that will select the basis functions and degree of the functions to be used for a simple global application of least-squares function approximation. Incorporation of this genetic algorithm also offers a fundamental advance. Previously, genetic algorithms used for function approximation optimized the weights or coefficients associated with a set of basis functions for a set of data. The basis functions themselves are selected *a priori*. The basis functions are often selected on a trial and error basis or as an educated decision given knowledge of the function. The genetic algorithm developed here sidesteps the guesswork involved by selecting the basis functions themselves as well as the degree of the basis functions. For example, say the following algebraic framework is the function approximation of some set of data:

$$\tilde{y} = c_1 p_1 + c_2 p_2 + c_3 p_3 + \dots \tag{1.2}$$

where c_i are the coefficients, and p_i are the basis functions. The genetic algorithm will test sets of basis functions, p_i, and use least-squares to solve for the coefficients, c_i. The genetic algorithm determines "fitness" of the selected basis functions by comparing the data set, y, with the approximation, $\tilde{y}$, using mean squared error (Eq.

1.3.

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\tilde{y}_i\right)^2 \tag{1.3}$$

Integrating the genetic algorithm into the learning algorithm, rather than focusing solely on post-processing of the data, allows the action-value function to be approximated on the side as the function evolves. This requires that the approximation maintains the relationship, though not necessarily the exact magnitude, between the numerical preferences for each action at a given state as encoded in the action-value function. The relationship between the preferences refers to the following: Given a state, $s \in S$, that has 5 actions associated with it, $(a_1, a_2, a_3, a_4, a_5) \in A$, and the following relationship between preferences

$$Q\left(s,a_3\right) > Q\left(s,a_1\right) > Q\left(s,a_4\right) > Q\left(s,a_2\right) > Q\left(s,a_5\right), \tag{1.4}$$

the approximation of the action-value function must at least preserve $Q\left(s,a_3\right)$ as the maximum preference for state s and preferably the full relationship of Eq. 1.4.

Other approximators, such as Global Local Orthogonal Polynomial Mapping (GLOMAP)[162], could be used in lieu of least-squares alone. GLOMAP allows for the blending or averaging of overlapping local approximations to yield a piecewise continuous model. These local approximations can be least-squares approximations with preselected basis functions, least-squares approximations with basis functions orthogonal to the GLOMAP weight functions, or any other local approximation. These characteristics make it a likely candidate as function approximator as the genetic algorithm presented here continues to be developed.

This genetic algorithm can be used separately as a standalone tool to approximate a given set of data, but for this dissertation it will be embedded in the learning algorithm such that the action-value function is approximated periodically as the

function evolves, thus reducing or eliminating post-processing and negating approximation issues of the often jagged final action-value function.

The approach is validated using computationally intensive models of a morphing airfoil and morphing wing representing the class of systems of interest, the simple classic dynamical system of the inverted pendulum, and thermal location for autonomous soaring. The dimensionality of the problems as quantified by the number of state-action pairs with and without the multi-resolution state-space discretization method is analyzed. This means comparing the number of state-action pairs using the multi-resolution method with the number that would result should the entire state-space be discretized at the finest level. Convergence to a useful policy is shown through Monte Carlo simulations, which refers to initializing the agent randomly in the state-space and letting it use the information contained in the policy to attempt to reach the goal as defined in that particular problem. A policy is considered "good" if the policy has converged to the point that the Monte Carlo simulations show the agent reaches the specified goal in at least 95% of the attempts. Convergence of the policy is also monitored by a direct policy comparison, where the policy is periodically extracted from the action-value function and compared to the most penultimate extracted policy. A stopping criterion is partially based on changes in the policy detected by this comparison method and partially on the Monte Carlo simulations, since the former just shows convergence to *a* policy and the latter shows whether or not the policy performs well. This is all conducted using both the policy extracted from the tabular action-value function and the approximated action-value function for comparison. The results using the approximated action-value function should be comparable to the tabular action-value function.

F. Organization

This dissertation is organized as follows. Chapter II gives an introduction to the reinforcement learning problem. It then describes the Q-learning algorithm, its derivation, proof of convergence, and considerations and issues. The next several chapters describe the various new components developed for the Q-learning algorithm. Chapter III details the general discretization method used as well as the development of the multi-resolution state-space discretization method. Next genetic algorithms are introduced in Chapter IV before describing in detail the genetic algorithm used for function approximation. A few simple examples are included to demonstrate the use of this genetic algorithm. Chapter V describes the final component, the policy comparison and performance evaluation method and its use as a stopping criterion. The full algorithm that includes the multi-resolution state-space discretization method, the function approximation genetic algorithm, and the policy comparison method is described in detail in Chapter VI. Flowcharts are included to illustrate and clarify how the components interact and form a single algorithm. In addition the algorithm developed is also compared to LSPI in this chapter to demonstrate how the new algorithm differs from the various least squares approaches. The algorithm is validated in a number of examples in the next four chapters. Chapter VII describes the development of the inverted pendulum model, its casting as a reinforcement learning problem, and results using the new algorithm. Chapters VIII, IX, and X do the same for the morphing airfoil, morphing wing, and autonomous soaring thermal location problems, respectively. Conclusions from the various examples are drawn in Chapter XI followed by recommendations for future work in Chapter XII.

CHAPTER II

REINFORCEMENT LEARNING

Interacting with our environment is one of the fundamental ways in which we learn. With no explicit teacher and only sensory inputs, we can learn much information about cause and effect. We seek to learn and understand how the environment responds to our actions. Reinforcement learning, a machine learning method, is a computational approach to learning from interaction. The reinforcement learning problem seeks to learn what to do so as to maximize numerical reward. Reinforcement learning, in effect, learns how to map situations, or states, to actions. The learner does this by exploring its environment and discovering which actions yield the most reward.

We also, in effect, learn many actions in a series of stages. The painter does not automatically try to create a masterpiece when first learning the art. The musician must learn scales before tackling a concerto. The aerospace engineer must first learn the basics of lift and drag before designing an aircraft. Much learning can be thought of as a process of learning the broader more general goal, and then learning and mastering the details by progressively finer degrees.

A. Introduction to Reinforcement Learning

Reinforcement learning is learning through interaction with the environment to achieve a goal. More specifically it is learning to map situations to actions to maximize some numerical reward. The learner or decision-maker is the *agent* and does not know what actions to take *a priori* as is common in most forms of machine learning. Everything outside of the agent comprises the *environment*. The agent's task is to learn a policy or control strategy for choosing actions that achieves its goals. To learn the correct

policy, which is a state to action mapping, $\pi : S \rightarrow A$, the agent receives a *reward*, or reinforcement, from the environment.[81]

The agent and environment interact continually is a series of discrete time steps, $t = 0, 1, 2, 3, \ldots$. At each time step t, a series of events occur. The agent first receives some representation of the environment's state, $s_t \in S$, where S is the set of all possible states. Based on the state, the agent chooses an action, $a_t \in A(s)$, where $A(s)$ is the set of actions available to the agent in state s_t. At the next time step, the agent receives a numerical reward, $r_{t+1} \in \Re$, and is in a new state, s_{t+1}.

As the agent moves from state to state selecting actions and receiving rewards, it generates a mapping, as stated earlier, of states to probabilities of selecting each possible action. This policy, $\pi_t(s, a)$, is the probability that $a_t = a$ at $s_t = s$.[81]

The agent seeks to maximize the reward it receives, or more formally, its expected return, R_t. The simplest form of R_t is the sum of the rewards received after time t as shown in Eq. 2.1.

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T \tag{2.1}$$

where T is the final time step, assuming there is a final step. This breakdown of a sequence into a finite number of steps is called an *episode*. Discounted return denotes the sum of discounted rewards the agent tries to maximize, Eq. 2.2

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

where γ is the discount rate and is $0 \leq \gamma \leq 1$. The discount rate effectively modulates the importance of future expected rewards. If $\gamma = 0$, the agent seeks only to maximize immediate rewards. As γ approaches 1, the agent takes future rewards into account more strongly.[81]

In this research, and for many reinforcement learning problems, it is assumed

that the problems can be modeled as Markov Decision Processes (MDPs) and cast in the reinforcment learning problem framework. An MDP satisfies the following conditional probability distribution function:

$$\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0\} = \\ = \Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{2.3}$$

for all $s_1, \ldots, s + t + 1$ and for all integers $t > 0$.[163] This means that rather than the transition to state s' and receiving reward r depending on all past states, actions, and rewards, the transition to state s' and receiving reward r is *only* dependent on s_t and a_t. A problem is considered an MDP if all the information necessary for the agent to make a decision is incorporated in the current state. The decision is not based on any past states visited, and is therefore path independent.

An underlying theme of almost all algorithms used to solve reinforcement learning problems is estimating *value functions*. A value function is a function of the state or of state-action pairs that estimates how good it is, in terms of future rewards, for the agent to be in a given state.[81] The value of a state s under some policy π is denoted $V^\pi(s)$ and is the expected return of starting in s and following π for all subsequent steps. This expectation is formalized in Eq. 2.4.

$$V^\pi(s) = E_\pi\{R_t | s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{2.4}$$

A similar relationship exists for action-value functions, $Q^\pi(s, a)$, which is defined as the value of taking action a in state s under policy π. This relationship is shown in Eq. 2.5.

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \tag{2.5}$$

These can be estimated from experience and usually satisfies some recursive relationship. This relationship for the value function is derived in Eq. 2.6 and holds for any policy π and any state s.

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t|s_t = s\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s\right\} \\ &= \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_{t+1} = s'\right\}\right] \\ &= \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right] \end{aligned} \tag{2.6}$$

where $\mathcal{P}^a_{ss'}$ is the probability of transition from state s to state s' under action a, and $\mathcal{R}^a_{ss'}$ is the expected immediate reward on transition from s to s' under action a. Eq. 2.6 is referred to as the Bellman Equation for V^{π}. The Bellman Equation for Q^{π} is

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\{R_t|s_t = s, a_t = a\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s, a_t = a\right\} \\ &= \sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma\sum_{a}\pi(s',a')E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_{t+1} = s', a_{t+1} = a'\right\}\right] \\ &= \sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma\sum_{a}\pi(s',a')Q^{\pi}(s',a')\right] \end{aligned} \tag{2.7}$$

The next logical step is to define the optimal value function and optimal action-value function. This starts with the assumption that there is an optimal policy, π^*, better than all the others. One policy is better than another if its expected return is

greater.[81] The optimal value function, V^*, is thus defined as

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.8}$$

for all $s \in S$. Similarly, the optimal action-value function is defined as

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{2.9}$$

for all $s \in S$ and $a \in A(s)$. Q^* can be written in terms of V^* because Eq. 2.9 is the expected return of taking action a in state s and following an optimal policy for all subsequent steps, see Eq. 2.10.

$$Q^*(s,a) = E_\pi\left\{r_{t+1} + \gamma V^*(s_{t+1}) \,|s_t = s, a_t = a\right\} \tag{2.10}$$

The related Bellman optimality equations are listed in Eq. 2.11 and 2.12.

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right] \tag{2.11}$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\right] \tag{2.12}$$

Ideally one would simply solve the set of linear Bellman optimality equations to acquire an optimal value function or optimal action-value function. However, that requires that the transition probabilities, $\mathcal{P}^a_{ss'}$, and expected immediate rewards, $\mathcal{R}^a_{ss'}$, be known.[81] That is often not the case, unfortunately, so other methods are often employed.

There are a number of ways to solve for the value or action-value functions. Three basic solution methods are Dynamic Programming (DP), Monte Carlo methods, and Temporal-Difference (TD) learning.[81]

DP refers to algorithms that computes optimal policies given a perfect model of the environment. These are often computationally expensive and depend on a

perfect model. DP algorithms include policy evaluation, policy improvement, policy iteration, value iteration, *etc.*[81]

Monte Carlo methods estimate value functions and try to find optimal policies. One advantage of these methods over basic DP is that they require only experience and not perfect knowledge of the environment. Learning can be conducted on-line or in simulation with no prior knowledge of environment dynamics. These methods learn based on an episode by episode averaging of sample returns. Monte Carlo methods include Monte Carlo policy evaluation, Monte Carlo estimation of action values, Monte Carlo control (both on-policy and off-policy), *etc.*[81]

TD learning can be thought of as a combination of ideas from both DP and Monte Carlo.[81] These methods learn from raw experience without the need for a model of the environment's dynamics. They bootstrap in the sense that estimates are updated *during* an episode based on other learned estimates. TD learning methods include TD prediction, Sarsa, *Q-learning*, actor-critic methods, *etc.*

B. Derivation of Q-Learning Algorithm

Q-learning, the central learning method of this research, is an off-policy TD control algorithm developed and popularized by Watkins in 1989.[161] Off-policy refers to the fact that this method estimates the value of a policy while using a *separate* policy for control whilst learning.

To derive the one-step Q-learning update equation used in this research, recall

Eq. 2.10, take its expectation, and expand.[164]

$$\begin{aligned} Q(s,a) &= E\left\{r_{t+1} + \gamma V^*(s_{t+1})\right\} \\ &= E\left\{r_{t+1}\right\} + \gamma E\left\{V^*(s_{t+1})\right\} \\ &= E\left\{r_{t+1}\right\} + \gamma \sum_{s'} \mathcal{P}^a_{ss'} V^*(s_{t+1}) \end{aligned} \tag{2.13}$$

Exploiting the close relationship between Q and V^*,

$$V^*(s) = \max_{a_{t+1}} Q(s, a_{t+1}), \tag{2.14}$$

Q can be expressed as

$$Q(s,a) = E\left\{r_{t+1}\right\} + \gamma \sum_{s'} \mathcal{P}^a_{ss'} \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \tag{2.15}$$

A decaying weighted average can now be taken of $Q(s,a)$ to ensure convergence as the function is updated. This training rule is the equation for one-step Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \tag{2.16}$$

where r is the *reward* or reinforcement received by the agent from the environment. The learning rate, α, moderates the degree to which an action-value is changed at each step. The discount factor, γ, determines the present value of future rewards, as stated previously. At each step the agent has a tuple, $(s_t, a_t, r_{t+1}, s_{t+1})$, with which to update the action-value function, $Q(s,a)$. The learned action value function, Q, directly approximates the optimal action-value function, Q^*, independent of the policy the agent follows. In procedural form the Q-Learning algorithm is as follows:

Q-Learning()

- Initialize $Q(s,a)$ arbitrarily
- Repeat (for each episode)

  - Initialize s
  - Repeat (for each step of the episode)
    * Choose a from s using policy derived from $Q(s,a)$ (e.g. ε-Greedy Policy)
    * Take action a, observe r, s'
    * $Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha\left[r_{t+1} + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right]$
    * $s \leftarrow s'$
  - until s is terminal
- return $Q(s,a)$

An episode consists of the agent beginning in state s, choosing action a, observing r and s' produced by the environment, and updating the action-value function. This process continues until the episode ends either by reaching a goal or after a predefined number of actions have been taken by the agent.

C. Exploration vs. Exploitation

A policy, π is the mapping of states to actions, $\pi : S \rightarrow A$, which means that the agent selects its next action a_t based on the current state s_t, or $\pi\left(s_t\right) = a_t$. When selecting the next action, one typical problem the agent has to face is the exploration-exploitation dilemma.[81] If the agent selects a greedy action that has the highest value, then it is exploiting its knowledge obtained so far about the values of the actions. If instead it selects one of the non-greedy actions, then it is exploring to improve its estimate of the non-greedy actions' values. Exploiting knowledge from the outset usually results in the agent finding and preferring local optima rather than the global goal.[81] Exploring from the outset and continuing throughout the learning

process, however, avoids this problem, though the agent is more likely to continue to randomly explore areas that are not of interest.[81] Given that Q-Learning is an off-policy reinforcement method, the policy followed while learning must be chosen to balance these concerns, promote efficient convergence, and tailored for the application of interest. Possible policies include greedy, non-greedy, ε-greedy (Eq. 2.17), and softmax action selections.

$$\begin{aligned} &\varepsilon - \text{greedy policy} \\ &\text{if(probability} > 1-\varepsilon) \\ &a = \arg\max_{a} Q(s,a) \\ &\text{else} \\ &a = rand(a_i) \end{aligned} \tag{2.17}$$

Many unique solutions exist to handle the exploration-exploitation policy dilemma of both discrete and continuous reinforcement learning problems. One such solution is to integrate the Metropolis Criterion into Q-learning, which eliminates some exploration blindness.[165, 166] A greedy exploration policy based on the state balance criterion can also be used.[167] Exploration and exploitation can also be decoupled to cope with any instabilities in the Q-learning algorithm.[168] An exploration algorithm that considers "prediction accuracy requirements" during exploration has been applied to a robot juggling Q-learning problem.[145] A Q-learning agent can also be restricted to explore only those options that are likely to avoid any unnecessary risk, which allows the algorithm to balance competing objectives and find satisficing solutions.[169] Similar to Goodrich's Satisficing Q-Learning is Park and Kim's two mode Q-learning, which also separates success and failure experiences to learn more quickly. Simsek and Barto propose a method efficient exploration aided by a principled heuristic.[170] Reference [171] describes a fairly extensive job of proving convergence for single-step on-policy reinforcement learning algorithms for control

with both decaying exploration and persistent exploration.

D. Proof of Convergence for Q-Learning

A proof that the Q-learning algorithm converges to the optimal action value function is described in detail by Watkins in Reference [161] and is summarized in this section.

From the Q-learning update in equation in Eq. 2.16, it can be seen that a set of tuples are used in each update. These tuples are the state, action taken, immediate reward received, and the next state, or $(s_t, a_t, r_{t+1}, s_{t+1})$, respectively. For the purposes of this section, these tuples will be denoted in the following manner, (s, a, r, s'). Each observation of a tuple is numbered $1, 2, 3, \ldots$ and are used to update Q. Therefore, let the n^{th} observation be (s_n, a_n, r_n, s'_n).

There are a number of observations that must now be made to aid in showing convergence. First, assume that the data available to the agent consists of an infinite sequence of observations. These observations are assumed to be independent observations of state transitions and rewards. It is also assumed that this is a fully Markov process. The observations do not need to be connected, so disconnected episodes of finite length can be used to collect the tuples.

Q is usually initialized to some finite values and is denoted $Q_0(s, a)$ before any updates are made. After the n^{th} observation is used to update Q, the action-value function is denoted $Q_n(s, a)$. The update equation using this notation is

$$Q_n(s,a) = \begin{cases} Q_{n-1}(s,a) + \alpha_n \left(r_n + \gamma \max_a Q_{n-1}(s,a) - Q_{n-1}(s,a) \right) \\ \qquad\qquad\qquad\qquad \text{if } s = s_n \text{ and } a = a_n \\ Q_{n-1}(s,a) \qquad\qquad\qquad\qquad \text{otherwise} \end{cases} \tag{2.18}$$

1. The Action-Replay Process

As did Watkins, now introduce the action-replay process (ARP), which is a hypothetical Markov decision process, to be used as a proof device.[161] The real process is abbreviated as RP. The ARP is constructed from the observations and consists of layers of states. In each layer, k, there is a state $< s, k >$ in the ARP corresponding to each state in the RP. The difference between the state s in the RP and the ARP is that the state s in the ARP has the same actions available to it, but the effects of those actions are different.

The effect of a selected action in the ARP is that the action is a "replay" of an observation. Performing an action a in state s is thus simulated by finding an observation of performing a in s, and the using the associated observed r and s' as the simulated reward and new state, respectively. The eligible observations for the simulation must occur before the current, or k^{th} observation. These observations are numbered n_1, n_2,..., n_i, where

$$n_1 < n_2 < \ldots < n_i \leq k. \tag{2.19}$$

Recall that α is the learning rate associated with how much Q is adjusted with each observation. With this in mind, the probability of replaying observation n_i is α_i, for n_{i-1} is $(1 - \alpha_i)\,\alpha_{i-1}$, *etc.* The ARP terminates with a final payoff of $Q_0(s, a)$ if no eligible action is selected for replay. The following is the pseudocode for this process:

- Perform a in $< s, 0 >$
  - Terminate the ARP with an immediate reward of $Q_0(s, a)$
  - Halt
- Perform a in $< s, k >$, for $k > 0$

- If $s = s_k$ and $a = a_k$
  - Begin
    - Either (with probability α_k
    - Go to $< s', k-1 >$ with an immediate reward of r_k
    - Halt
    - Or (with probability $1 - \alpha_k$
    - Perform a in $< s, k-1 >$
  - End
- Else
  - Perform a in $< s, k-1 >$
- End

Note that it is not possible to perform an infinite sequence of actions in the ARP. Eventually the process will reach level 0 and terminate.

Now it can easily be shown that Q_n defines the optimal action-value function for the ARP at stage n.[161] Let $Q^*_{ARP}(< s, n >, a)$ be the optimal action-value function for the ARP for action a at state $< s, n >$. Let V^*_{ARP} be the optimal value function for the ARP.

Theorem D.1. The 'Action-Replay' Theorem *For all s, a, $Q_n(s,a)$ is the optimal action value at stage n of the ARP. That is*

$$Q_n(s,a) = Q^*_{ARP}(< s, n >, a) \tag{2.20}$$

for all s, a, and for all $n \geq 0$.

Proof. From the construction of the ARP, $Q_0(s,a)$ is the optimal and only possible

action value of $< a, 0 >$, a. Therefore,

$$Q_0(s,a) = Q^*_{ARP}(< s, 0 >, a) \tag{2.21}$$

By induction, the theorem holds for $n = 0$.

Suppose that the values of Q_{n-1} produced by the update rule in Eq. 2.16 are the optimal action-values for the ARP at stage $n-1$,

$$Q_{n-1}(s,a) = Q^*_{ARP}(< s, n-1 >, a), \tag{2.22}$$

for all s, a.

From Eq. 2.18, recall that $Q_n(s,a) = Q_{n-1}(s,a)$ for all s, a not equal to s_n, a_n, and

$$Q_n(s_n,a_n) = Q_{n-1}(s_n,a_n) + \alpha_n\left(r_n + \gamma \max_a Q_{n-1}(s_{,n},a) - Q_{n-1}(s_n,a_n)\right) \tag{2.23}$$

As a result, for all s, a not equal to s_n, a_n, performing a in $< s, a >$ in the ARP gives the same result as performing a in $< s, n-1 >$. Thus,

$$Q^*_{ARP}(< s, n >, a) = Q^*_{ARP}(< s, n-1 >, a) \tag{2.24}$$

for s, a not equal to s_n, a_n. Therefore,

$$Q^*_{ARP}(< s, n >, a) = Q_n(s,a) \tag{2.25}$$

for *all* s, a not equal to s_n, a_n.

Next, consider the optimal action-value of $< s_n, n >$, a_n in the ARP. Performing a in $< s_n, n >$ has two possible effects:

1. With probability α_n
    - Yield immediate reward r_n and new state $< s'_n, n-1 >$, or

2. With probability $1-\alpha_n$

   - The same effect as performing a_n in $< s, n-1 >$.

The optimal action-value in the ARP of $< s_n, a_n >$ is therefore

$$\begin{aligned} Q^*_{ARP}\left(< s_n, n >, a_n\right) &= \left(1-\alpha_n\right) Q^*_{ARP}\left(< s_n, n-1 >, a_n\right) + \\ &+ \alpha_n \left(r_n + \gamma \max_a Q^*_{ARP}\left(< s'_n, n-1 >, a\right)\right) \\ &= Q_{n-1}\left(s_n, a_n\right) + \\ &+ \alpha_n \left(r_n + \gamma \max_a Q_{n-1}\left(s'_n, a\right) - Q_{n-1}\left(s_n, a_n\right)\right) \\ &= Q_n\left(s_n, a_n\right) \end{aligned} \tag{2.26}$$

Therefore, by induction

$$Q_n\left(s_n, a_n\right) = Q^*_{ARP}\left(< s_n, n >, a_n\right) \tag{2.27}$$

for all s, a, which was to be proved. □

2. Convergence of Q^*_{ARP} to Q^*

The sufficient conditions under which the optimal action-values for the ARP at the n^{th} stage will converge to the optimal action values for the real process as $n \to \infty$ are the following.[161]

- There are an infinite number of observations in the form of tuples, (s, a, r_n, s'_n).
- The learning rate α_n for the observations are positive, decrease monotonically as n increases, and tend to 0 as $n \to \infty$.
- The sum of the learning rates for the observations is infinite.

To demonstrate the sufficiency of these conditions, first let the sum of the learning rates for all of the observations, (s, a, r_l, s'_l) with $l \leq k$ be the depth of a state-action

pair, $d(< s, k >, a)$ in the replay process. Following the replay process procedure outlined above for performing action a in $< s, k >$, the probability of reaching $< s, 0 >$ becomes arbitrarily small as the depth, $d(< s, k >, a)$, becomes large.

Recall that there are a finite number of state-action pairs. According to the third condition above, $d(< s, k >, a) \rightarrow \infty$ as $n \rightarrow \infty$. For any given D or ε, n can be chosen such that

$$\max_{m>n} (\alpha_m) < \varepsilon. \tag{2.28}$$

Given such an n, n' can then be chosen such that

$$\min_{s,a} (d(< s, n' >, a) - d(< s, n >, a)) > D \tag{2.29}$$

It is then possible to choose a sequence of values n_1, n_2, n_3, n_4, ... for any D and any ε such that the depths of *all* state-action pairs increase by D between each value of the sequence. As a result n can be chosen so large that the possible number of replayed observations is larger than any chosen k with probability as close to 1 as needed or desired. This also means that the largest learning rate α is so small that the transition probabilities and average rewards of the ARP are uniformly as close to those of the RP as desired. Thus, if n is sufficiently large, then

$$Q^*_{ARP}(< s, n >, a) \rightarrow Q^*(s, a) \tag{2.30}$$

with probability as close to 1 as desired.

CHAPTER III

STATE-SPACE DISCRETIZATION OF A CONTINUOUS DOMAIN FOR Q-LEARNING

A. Problem Definition

Q-learning on a continuous domain quickly becomes intractable when one considers that convergence of the algorithm to the optimal action-value function is only guaranteed if the agent visits every possible state an infinite number of times.[161] An agent would therefore visit an infinite number of states using an infinite number of actions an infinite number of times. Add in the fact that the states can be defined by anywhere from 1 to N continuous variables and the dimensionality of the problem becomes a significant issue.

B. Learning on a 2-, 4-, and N-Dimensional Domain

One way to cope with the inherent complexity of a continuous domain learning problem is to discretize the state-space by overlaying a pseudo-grid. The essential ideas of this concept can be best introduced in terms of a 1-dimensional problem. The notation can then be generalized for the 2-, 4-, and N-dimensional problems.

For the 1-dimensional problem the state-space can be represented by a line as seen in Figure 1. An arbitrary set of vertices $\left\{{}^{1}X, {}^{2}X, \ldots, {}^{k}X, \ldots\right\}$ are introduced at a uniform distance h apart. Ideally, h is chosen such that a vertex lies on both end points of the state-space. In the learning algorithm the agent is either only allowed to visit the overlaying vertices and their corresponding states or the states are quantized such that information from surrounding states are stored in the nearest vertex. This technique effectively reduces the state-space from infinity to a finite number of states,

thus rendering the problem more manageable.

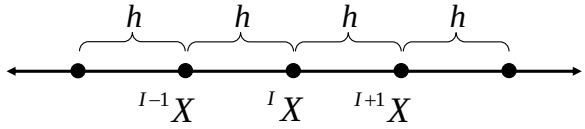

Fig. 1. 1-Dimensional State-Space with Overlaying Pseudogrid

To further simplify the problem, we restrict what actions the agent may take. When the agent is at the I^{th} vertex $X = {}^{I}X$, it may only move to ${}^{I-1}X$ or ${}^{I+1}X$. Now the problem only has two possible actions rather than an infinite number, which further reduces the problem complexity.

Let L denote the length of the continuous domain. As per our formulation there are

$$N_{V_1} = \frac{L}{h} + 1 \tag{3.1}$$

vertices, where N_V is the number of vertices, and 2 actions. Therefore, there are only

$$N_1 = 2\left(\frac{L}{h} + 1\right) \tag{3.2}$$

state-action pairs, where N is the number of state-action pairs.

The 2-dimensional problem can be represented in a similar manner. In this case the state-space is represented by Figure 2. An arbitrary set of vertices $\{{}^{11}X, {}^{12}X, \ldots, {}^{ij}X, \ldots\}$ are again introduced at uniform distances h_{x_1} or h_{x_2} apart. The actions available to the agent are again restricted as in the 1-dimensional case. For the 2-dimensional case, when the agent is at the IJ^{th} vertex $X = {}^{IJ}X$, it may only move to vertices ${}^{(I-1)J}X$, ${}^{(I+1)J}X$, ${}^{I(J-1)}X$, and ${}^{I(J+1)}X$, a total of 4, or $2 * 2$, actions.

This problem is more complex than the previous one, yet it is still simpler than a 2-dimensional continuous state-space problem. For this 2-dimensional discrete case,

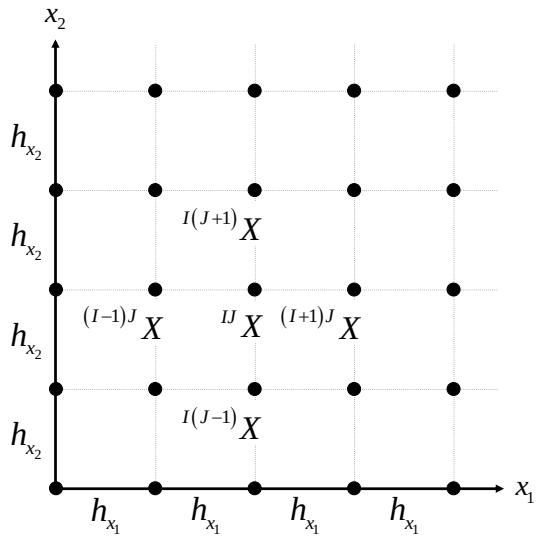

Fig. 2. 2-Dimensional State-Space with Overlaying Pseudogrid

let L_{x_1} and L_{x_2} denote the length in the x_1- and x_2-direction, respectively, of the continuous domain. This results in

$$\begin{aligned} N_{V_2} &= \left(\frac{L_{x_1}}{h_{x_1}} + 1\right)\left(\frac{L_{x_2}}{h_{x_2}} + 1\right) \\ &= \prod_{i=1}^{2}\left(\frac{L_{x_i}}{h_{x_i}} + 1\right) \end{aligned} \tag{3.3}$$

vertices. Therefore, there are

$$N_2 = 2 * 2\prod_{i=1}^{2}\left(\frac{L_{x_i}}{h_{x_i}} + 1\right) \tag{3.4}$$

state-action pairs.

The 4-dimensional problem is a simple extension of the 1- and 2-dimensional problems. In the 4-dimensional case the state-space is represented by Figure 3. An arbitrary set of vertices $\left\{{}^{1111}X, {}^{1112}X, \ldots, {}^{ijkl}X, \ldots\right\}$ are again introduced at uniform distances h_{x_1}, h_{x_2}, h_{x_3}, or h_{x_4} apart. The actions available to the agent are again

restricted as in the 1- and 2-dimensional case. For the 4-dimensional case, when the agent is at the $IJKL^{th}$ vertex $X = {}^{IJKL}X$, it may only move to vertices ${}^{(I-1)JKL}X$, ${}^{(I+1)JKL}X$, ${}^{I(J-1)KL}X$, ${}^{I(J+1)KL}X$, ${}^{IJ(K-1)L}X$, ${}^{IJ(K+1)L}X$, ${}^{IJK(L-1)}X$, and ${}^{IJK(L+1)}X$, a total of 8, or $2 * 4$, actions.

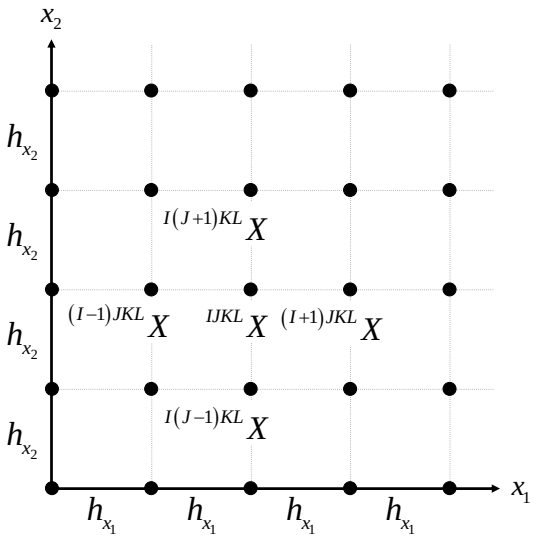

Fig. 3. 4-Dimensional State-Space with Overlaying Pseudogrid

Similar to the simpler cases, for the 4-dimensional discrete case, let L_{x_1}, L_{x_2}, L_{x_3}, and L_{x_4} denote the length in the x_1-, x_2-, x_3-, and x_4-direction, respectively, of the continuous domain. This results in

$$\begin{aligned} N_{V_4} &= \left(\frac{L_{x_1}}{h_{x_1}} + 1\right)\left(\frac{L_{x_2}}{h_{x_2}} + 1\right)\left(\frac{L_{x_3}}{h_{x_3}} + 1\right)\left(\frac{L_{x_4}}{h_{x_4}} + 1\right) \\ &= \prod_{i=1}^{4}\left(\frac{L_{x_i}}{h_{x_i}} + 1\right) \end{aligned} \tag{3.5}$$

vertices, which means there are

$$N_4 = 2 * 4\prod_{i=1}^{4}\left(\frac{L_{x_i}}{h_{x_i}} + 1\right) \tag{3.6}$$

state-action pairs.

From here the formulation can be generalized to the N-dimensional case. For an N-dimensional continuous state-space, an arbitrary set of vertices $\left\{{}^{11\ldots1}X, {}^{11\ldots2}X, \ldots, {}^{NN\ldots N}X\right\}$ are introduced at uniform distances h_{x_1}, h_{x_2}, ..., h_{x_N} apart. The actions are restricted to the two nearest vertices in any direction from the current vertex $X = {}^{IJ\cdots}X$, yielding a total of $2N$ actions available to the agent from any given vertex.

Now let L_{x_1}, L_{x_2}, ..., L_{x_N} denote the length in the x_1-, x_2-, ..., and x_N-directions, respectively. This formulation leads to

$$N_{V_N} = \prod_{i=1}^{N} \left(\frac{L_{x_i}}{h_{x_i}} + 1 \right) \tag{3.7}$$

vertices and

$$N_N = 2N \prod_{i=1}^{N} \left(\frac{L_{x_i}}{h_{x_i}} + 1 \right) \tag{3.8}$$

state-action pairs.

Intuitively, the larger h_{x_i} is, the fewer the number of vertices, resulting in fewer visits by the agent necessary to learn the policy correctly. Special care must be taken, however, in the choice of h_{x_i} and the definition of the goal the agent attempts to attain. If the only goal state lies between vertices or is so specific the agent is unlikely to find it and store that learned information in a quantized state, then the agent will be unable to learn the actions necessary to reach the goal state.

Discretizing the domain in this way can greatly simplify a learning problem, though limiting the action in this way is a significant assumption. In many applications, such as a robotics application, an action can change multiple state variables. Limiting the actions simplifies the reinforcement learning problem so that focus during the early development and validation of this algorithm is on the algorithm itself rather than the complexity of the application. Future development of the algorithm,

however, should eliminate this assumption.

C. Multi-Resolution State-Space Discretization (AAG) for N-Dimensions

Discretizing a state-space for learning is beneficial in that it creates a finite number of state-action pairs the agent must visit. Generally, as the number of state-action pairs decreases, the rate of convergence increases.[172] However, fewer state-action pairs capture less detail of the environment. Also, using the method described in Section B limits the agent to storing information only at the vertices. It is entirely possible that the goal the agent is seeking, or any other area of interest, does not lie on a vertex. This necessitates adding a range to the goal that encompasses one or more of the vertices in the state-space. These vertices within the goal range are defined as pseudo-goals. (Figure 4)

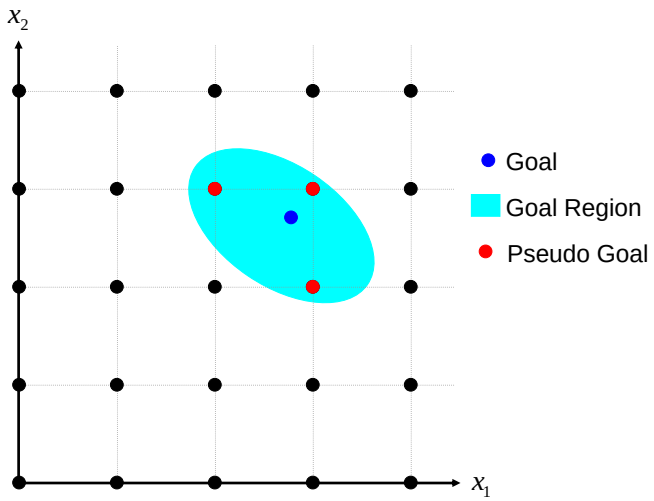

Fig. 4. Multi-Resolution State-Space Discretization – Phase 1: Coarse Grid, Large Goal Range

As the agent explores the coarsely discretized state-space and garners rewards,

it also notes the location of the pseudo-goals. Once learning on the current discretization has converged, the area surrounding and encompassing the psuedo-goals is rediscretized to a finer resolution such that $h_{x_{i_2}} < h_{x_{i_1}}$, where the subscript 1 denotes the initial discretization, and subscript 2 denotes the second discretization. A new, smaller range is defined for the goal and learning begins anew in the smaller state-space. The agent retains the information already stored in the action-value function and simply continues to add to it as it learns the smaller region in more detail. If no pseudo-goals are found at the current level of discretization, then learning does not continue at a finer level of discretization. It is is known that a goal exists in the state-space and is not found when learning on the coarsest discretization, then parameters such as the initial discretization and goal range must be adjusted.

Figure 5 shows the re-discretization of the state-space.

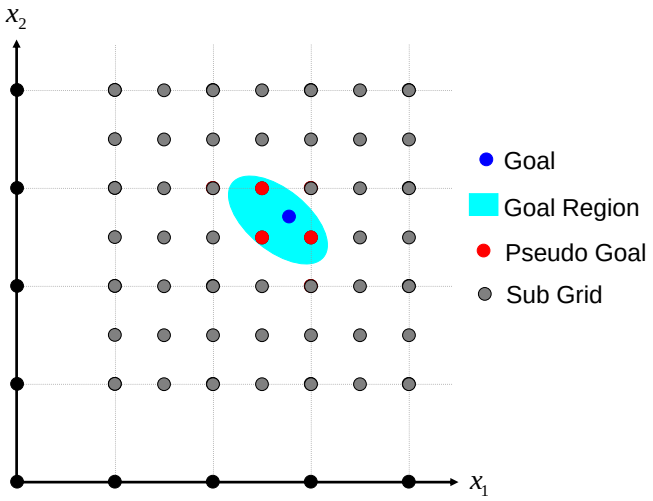

Fig. 5. Multi-Resolution State-Space Discretization – Phase 2: Finer Grid, Smaller Goal Range

This method can then be generalized for the N-dimensional case. Let $L^j_{x_1}$, $L^j_{x_2}$, ..., $L^j_{x_N}$ denote the length in the x_1-, x_2-, ..., and x_N-directions, respectively, and

the superscript j denote the resolution of the discretization in which 1 is the coarsest and M is the finest. The vertices for each resolution are then set at distances $h^j_{x_1}$, $h^j_{x_2}$, ..., $h^j_{x_N}$ apart. These terms effectively define the fineness of resolution level j. Eqs. 3.7 and 3.8 can then be modified to calculate the number of vertices and state action pairs for this method, as shown in Eqs. 3.9 and 3.10.

When the multi-resolution learning is complete, there are

$$N_{V_N} = \sum_{j=1}^{M} \left(\prod_{i=1}^{N} \left(\frac{L^j_{x_i}}{h^j_{x_i}} + 1 \right) \right) - \sum_{j=1}^{M-1} \left(\prod_{i=1}^{N} \left(\frac{L^{j+1}_{x_i}}{h^j_{x_i}} + 1 \right) \right) \tag{3.9}$$

vertices. Therefore, there are

$$N_N = 2N \left(\sum_{j=1}^{M} \left(\prod_{i=1}^{N} \left(\frac{L^j_{x_i}}{h^j_{x_i}} + 1 \right) \right) - \sum_{j=1}^{M-1} \left(\prod_{i=1}^{N} \left(\frac{L^{j+1}_{x_i}}{h^j_{x_i}} + 1 \right) \right) \right) \tag{3.10}$$

state-action pairs. Notice the second term of each equation excises the duplicate vertices from one level of discretization to the next. Also note that if the full state-space were simply discretized by the finest level of $h^M_{x_i}$, there would be

$$N_{V_{N_{fine}}} = \prod_{i=1}^{N} \left(\frac{L^1_{x_i}}{h^M_{x_i}} + 1 \right) \tag{3.11}$$

vertices and

$$N_{N_{fine}} = 2N \left(\prod_{i=1}^{N} \left(\frac{L^1_{x_i}}{h^M_{x_i}} + 1 \right) \right) \tag{3.12}$$

state-action pairs. It can be shown that $N_{V_N} < N_{V_{N_{fine}}}$ and $N_N < N_{N_{fine}}$ by a significant amount, the magnitude of which is determined by the factor by which each subsequent discretization is reduced from the previous.

It is known that the time to convergence for Q-learning increases exponentially as the complexity of the problem, e.g. the quantity of state-action pairs, increases. This method reduces a learning problem to a series of smaller learning problems with relatively few state-action pairs, on the order of several orders of magnitude less.

Rather than one large problem that could take a great deal of time to converge, there are several quickly converging smaller problems.

D. Multiple Goal Regions

In many problems it is entirely possible that more than one distinct area of the state-space contains pseudo-goals. When rediscretizing the state-space from level j to $j+1$ using the multi-resolution discretization method described above, one option is simply to have the algorithm define $L_{x_1}^{j+1}$, $L_{x_2}^{j+1}$, ..., $L_{x_N}^{j+1}$ such that the region encompasses *all* of the pseudo-goals. The drawback of this approach is that if there are distinct groupings of pseudo-goals, then this approach discretizes to a finer degree regions that contain nothing of interest to the agent. Continuing to learn in these regions is essentially wasted effort.

Another approach is to separate and discretize each grouping of pseudo-goals individually. Using the previous notation, this approach would essentially yield $L_{x_1}^{j_1+1}$, $L_{x_2}^{j_1+1}$, ..., $L_{x_N}^{j_1+1}$ for one group of pseudo-goals as denoted by the j_1, $L_{x_1}^{j_2+1}$, $L_{x_2}^{j_2+1}$, ..., $L_{x_N}^{j_2+1}$ for the second group, and $L_{x_1}^{j_n+1}$, $L_{x_2}^{j_n+1}$, ..., $L_{x_N}^{j_n+1}$ for the n^{th} distinct group of pseudo-goals.

The set of pseudo-goals recorded by the agent are separated into their groups using a general clustering technique from the statistical analysis toolbox in MATLAB$^{\text{TM}}$. Let R denote the recorded pseudo-goals to be clustered. R is an $m \times n$ matrix with m observations of n variables, or m pseudo-goals defined by n state variables in the context of machine learning. The first step in the clustering process is to determine the pairwise distance of all of the observations using the following equation for the Euclidean distance:

$$d_{rt}^2 = (\mathbf{x}_r - \mathbf{x}_t)(\mathbf{x}_r - \mathbf{x}_t)' \tag{3.13}$$

When arranged into a row vector, $\mathbf{y}$, this yields a vector length of $m(m-1)/2$ arranged in the order $[(2,1),(2,1),\ldots,(n,1),(3,2),\ldots,(n,2),\ldots,(n,n-1)]$.

A hierarchical cluster tree, Z, where Z is an $(m-1)\times 3$ matrix, is then created from the Euclidean distances encoded in $\mathbf{y}$. Cluster indices linked in pairs are denoted in columns 1 and 2 of Z to form a binary tree. Leaf nodes, or the singleton clusters from which all higher clusters are built, are numbered from 1 to m and are the original observations. Column 3 contains the linkage distance, as calculated by Eq. 3.14, between two merged clusters. When a new cluster is formed, a row is added to Z and is assigned the index $m+I$, where $Z(I,1:2)$ contains the clusters indices that form cluster $m+I$ and $Z(I,3)$ contains the linkage distance between the two.

$$d(r,t) = \min\left(\left(\mathbf{x}_{r_i} - \mathbf{x}_{t_j}\right)\left(\mathbf{x}_{r_i} - \mathbf{x}_{t_j}\right)'\right), i \in (1,\ldots,n_r), j \in (1,\ldots,n_t) \qquad (3.14)$$

Clusters are then formed and assigned cluster numbers from these linkages by comparing the length of each link in Z with the average length at the same level in the hierarchy.

Each cluster represents a distinct set of pseudo-goals contained in R. Using these clusters, the multi-resolution discretization method can now be applied to each individual cluster in turn, allowing the agent to learn each region in finer detail and neglect the uninteresting regions in between.

CHAPTER IV

FUNCTION APPROXIMATION

Practical reinforcement learning problems often have continuous state and action spaces. Many reinforcement learning methods, however, assume finite state and action spaces. In terms of Q-Learning, learning on continuous state and action spaces translates into an infinite number of possible state-action pairs. A solution is to discretize the state and action spaces and approximate the action-value function for the entire space after learning has occurred. Approximation is in itself an issue to be addressed. Generally, function approximation is performed on the final function with methods such as GLOMAP[162], SFA[173, 174], linear least-squares, *etc.* The user must often choose the set of basis functions and make a decision as to the degree of the functions. These choices are often made on a trial and error basis or as an educated decision given knowledge of the function. Indeed, there tends to be some guesswork involved.

Learning based approaches to value and action-value function approximation include Parr *et al.*'s Bellman-error-based approach to generating orthogonal basis functions[175], Lee's fuzzy model with a genetic algorithm and particle swarm optimization to tune model precision[176], and Hauser's genetic algorithm that determines a piece-wise polynomial approximation with integer coefficients[177].

A. Introduction to Genetic Algorithms

A genetic algorithm (GA) is considered an *evolutionary* method because it mimics or is analogous to the way biological organisms develop and evolve skilled behavior that they do not actually learn during their lifetime.[81] GAs use operators similar to those found in nature to create successive generations of hypotheses for the problem

at hand from an initial set of hypotheses by mutating and recombining parts of the best currently known hypotheses. According to Mitchell the popularity of GAs is motivated by the following three factors, though there are many others:[164]

- "Evolution is known to be a successful, robust method for adaptation within biological systems."
- "GAs can search spaces of hypotheses containing complex interacting parts, where the impact of each part on overall hypothesis fitness may be difficult to model."
- "Genetic algorithms are easily parallelized and can take advantage of the decreasing costs of powerful computer hardware."

GAs begin with an initial set of hypotheses called a *population*. Each hypothesis is represented by a string of numbers. This string can be a row vector of real numbers, such as for the problem of finding the minimum of the function $z = ax + by$ in which $[x\ y]$ is a hypothesis, or a bit string of the form 011010010111. Designing a bit string to represent a hypothesis often requires creativity and finesse.

The *fitness* of each hypothesis in the current population is them evaluated. Fitness is a measure of "how good" the hypothesis is. Fitness can be the accuracy of a hypothesis with respect to some training data, the number of games won in checkers or chess when playing against other hypotheses in the current population, *etc.*

The successor population is created next. A portion of the hypotheses of the successor population are selected for inclusion as is from the current population. These are called "elite children". The one or two that have the best fitness are usually selected automatically. The rest to be included can be selected in a number of ways including:[164]

- *probabilistically* - the probability of selecting h_i given by

$$\Pr(h_i) \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)} \tag{4.1}$$

  where p is the number of hypotheses in the population,

- *tournament selection* - two hypotheses are chosen at random from the current population, and the more fit of the two is selected with probability p while the less fit is selected with probability $(1 - p)$, and

- *rank selection* - the hypotheses are sorted by fitness, and the probability a hypothesis is selected is based on its rank in this list.

The rest of the successor population is generated using two operators: crossover and mutation. The *crossover fraction* is the fraction of the next generation that are produced by crossover, other than the elite children. The rest are created by mutation. The *crossover operator* produces two new offspring or *children* from two *parents*. The children are created by copying bits from each of the parents. Figure 6 illustrates the mutation operator and the three types of crossover: single-point, two-point, and uniform. The *mutation operator* creates a child from a single parent by randomly choosing a single bit and changing its value.[164]

The parent strings on which the two operators are applied are selected in a manner similar to those selected as elite children. There are five predefined methods in MATLAB™ for choosing these parents:

- *Stochastic uniform* - creates a line in which each possible parent corresponds to a section of the line proportional to its fitness. Randomly sized steps are taken along the line and the parent from the section of the line at each step is selected.

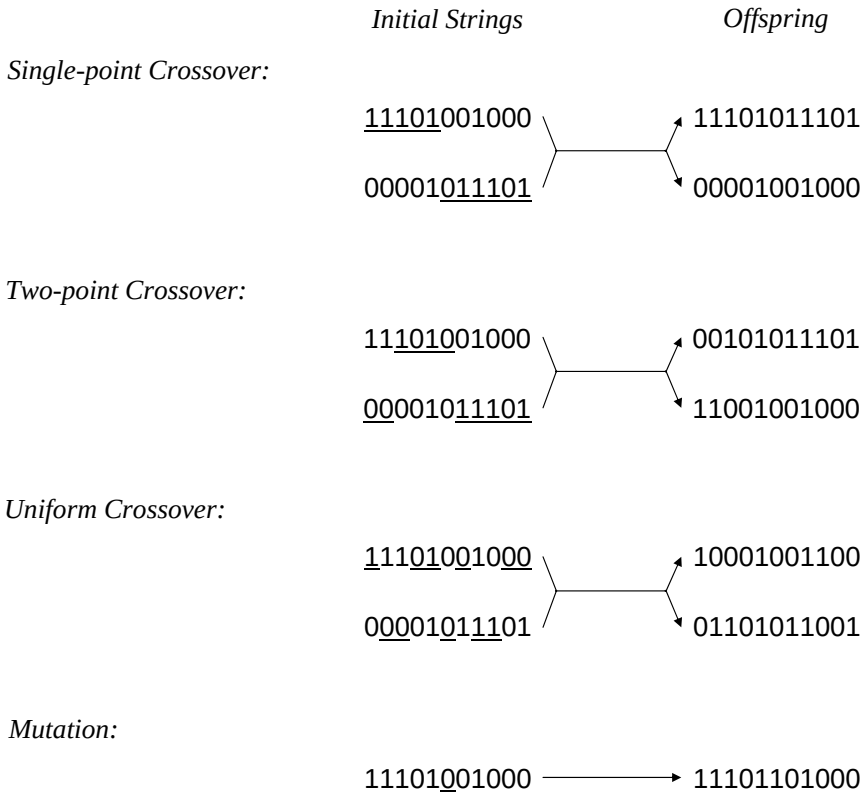

Fig. 6. Common Genetic Algorithm Operators

- *Remainder* - based on the scaled value of the fitness for each parent.
- *Uniform* - based on the fitness of the parent and the total number of possible parents.
- *Roulette* - creates a "roulette wheel" in which the area allocated to each parent is based on the fitness of the parent. A random number is used to select one of the sections with a probability equal to its area.
- *Tournament* - two or more parents are chosen at random, and the parent with the highest fitness is selected.

This process of creating a new generation and evaluating fitness continues until either the user specified total number of generations is reached, optimization stalls in the sense that there is no longer any significant change in the population, or a time limits is reached. It is possible that an optimal or near optimal solution is not achieved. Occasionally, a GA can get "stuck" at a local extrema. This problem and many others can be avoided by modifying the many variables of the GA, such as the population size, the range of hypotheses in the initial population, the crossover fraction, the various selection functions, *etc.* These must usually be tailored or tuned to each particular optimization problem.

B. Genetic Algorithm for Function Approximation

The GA used in this research is applied to the problem of function approximation, more specifically, approximation of the action-value function, though it can be used for function approximation of *any* data set. Usually, when one thinks of optimizing or solving an approximation of the form

$$\tilde{\mathbf{y}} = \phi(\mathbf{x})\,\omega \tag{4.2}$$

where $\tilde{\mathbf{y}}$ is the approximated y-values and is an $m \times 1$ vector, $\phi(\mathbf{x})$ are basis functions that are a function of $\mathbf{x}$ and is an $m \times n$ matrix, and ω are the weights applied to each basis function and is an $n \times 1$ vector, then the hypotheses of the GA represent the sets of weights. The basis functions are therefore selected *a priori*. A possible drawback with this approach is that the success of the GA is dependent on the selected basis functions. Some knowledge of the data to be approximated is often necessary to choose good basis functions initially, otherwise trial and error of basis functions sets becomes necessary.

The GA developed here takes just the opposite approach. Rather than searching over the set of weights, this GA searches over the set of *basis functions* and uses least-squares to calculate weights. Instead of the user essentially guessing basis functions and the GA choosing weights, only the basis functions are manipulated. The major responsibility of the user reduces to providing an adequate number of basis functions sets for the GA to search over and to choose a method to calculate the weights.

The population of this GA is comprised of bit strings that designate a numeric value that is assigned to each set of basis functions and a numeric value corresponding to the degree of functions, or the number of nodes when appropriate. This is shown in Eq. 4.3.

$$0000010 \tag{4.3}$$

where the first three bits comprise the gene representing the basis function set and the last four bits comprise the gene representing the degree or number of nodes or centers. Let the two genes be numbers of base 2 and let $000 = 0$ be assigned to basic polynomials, then the bit string in Eq. 4.3 denotes 2^{nd} degree polynomial basis functions. Other parameters for the GA developed for this research are listed in Table I.

The chosen method of calculating the weights is linear least-squares. Linear least-squares calculates the optimal estimate for the function approximation using the following equation.

$$\omega = \left(\phi^T(\mathbf{x})\,\phi(\mathbf{x})\right)^{-1}\phi^T(\mathbf{x})\,\mathbf{y} \tag{4.4}$$

where $\mathbf{y}$ are the measured y-values. This formulation requires that there are at least n independent observation equations, so there must be a linearly independent basis function for every weight, and $\phi^T(\mathbf{x})\,\phi(\mathbf{x})$ is *strictly* positive and invertible. For a full derivation and discussion, please consult Reference [178].

Table I. GA for Function Approximation Parameters Settings

GA Parameter	Value
Population Type	Bit String
Population Size	40
Elite Count	2
Crossover Fraction	0.8
Generations Limit	100
Time Limit	∞
Initial Population	–
Initial Fitness Scores	–
Creation Function	Uniform
Fitness Scaling Function	Rank
Selection Function	Stochastic Uniform
Crossover Selection Function	Scattered
Mutation Rate	0.01

Linear least-squares is used instead of more sophisticated approximators such as GLOMAP or SFA for a number of reasons. The first is ease of use. Eq. 4.4 is all that is needed to calculate the weights given that $\phi(\mathbf{x})$ and $\phi^T(\mathbf{x})\,\phi(\mathbf{x})$ meet requirements. Selecting and interchanging basis functions is a simple matter of modifying ϕ and can be easily manipulated by the GA. Linear least-squares can handle scattered data with ease as well. Early incarnations of GLOMAP require careful node location selection and a uniform distribution of data throughout the space to be approximated, which might not be available for an action-value function. Other methods tend to require tuning of their parameters. Tuning of parameters is not possible while the GA is churning through generations of hypotheses. A drawback to linear least-squares is that it is a global approximation of the data set. Small localized features tend to be smoothed over. GLOMAP and others are much more adept at capturing local detail. This problem can be slightly alleviated by providing more observations in the area with local phenomena, which is exactly what the multi-resolution method developed in Chapter III does, to "encourage" weights that will more accurately represent those areas.

1. Basis Function Selection

As stated previously, the first gene of the bit string searched by the GA describes the set of basis functions to be used for function approximation. The selection of basis functions available to the GA are preset by the user. This is the only instance in which the user has any direct effect on the basis functions. The basis functions and their associated bit string gene used in this research are listed in Table II.

The second gene for those bit strings encoding for polynomials or trigonometric polynomials describe the degree of the polynomial. The degree ranges from 0 to 15. This range results in 3 coefficients for a 2^{nd} degree polynomial with one independent

Table II. GA for Function Approximation Basis Function Sets

Basis Function	1^{st} Gene	Basis Equation
Polynomials	000	$y = \omega_0 + \sum_{n=1}^{N} \omega_n x^n$
Trigonometric Polynomials	001	$y = a_0 + \sum_{n=1}^{N} a_n \cos(nx) + \sum_{n=1}^{N} b_n \sin(nx)$
Linear Interpolation	010	$y = mx + b$
Spline Interpolation	011	$S(x) = \begin{cases} S_0(s), x \in [x_0, x_1] \\ S_1(s), x \in [x_1, x_2] \\ \cdots \\ S_{n-1}(s), x \in [x_{n-1}, x_n] \end{cases}$ where $S(x_i) = y(x_i)$, $S_{i-1}(x_i) = S_i(x_i), i = 1, \ldots, n-1$, $S'_{i-1}(x_i) = S'_i(x_i), i = 1, \ldots, n-1$, $S''_{i-1}(x_i) = S''_i(x_i), i = 1, \ldots, n-1$
Linear Radial Basis Functions (RBFs)	100	$\phi = -(wr)$
Cubic RBFs	101	$\phi = (wr)^3$
Gaussian RBFs	110	$\phi = \exp\left(\frac{(wr)^2}{\sigma^2}\right), \sigma = 0.5$
Multiquadratic RBFs	111	$\phi = \sqrt{(wr)^2 + \sigma^2}, \sigma = 0.5$

variable, 6 coefficients for a 2^{nd} degree polynomial with two independent variables, *etc.* A similar relationship holds for the trigonometric polynomials.

The interpretation of the second gene is slightly different for the linear interpolation and spline interpolation. These two sets use the MATLAB^{TM} function "interpn". The second gene defines the *number of knots* to be created as input to the m-file. Let b_n be the base 10 number encoded by the second gene, and let s_n be the number of state or independent variables. The number of knots, n_n, is calculated by

$$n_n = (b_n + 2)^{s_n} \tag{4.5}$$

and are set uniformly throughout the space, including the edges of the space. The $\tilde{y}$-values for each knot are taken from the data set to be approximated. If there is no data entry corresponding to the knot, then the y-value for the data entry *nearest* to the knot is used. It is these knots and y-values that are used as the approximation rather than using linear least-squares to calculate weights.

The radial basis functions are distributed through the space in a similar manner, but instead of knots, it is their centers, c, that are distributed such that $r = \|\mathbf{x} - \mathbf{c}\|$. The w in each RBF equation is a simple scaling factor dependent on the dimensions of the space defined by the data set. Let ${}^1\mathbf{x}$ and ${}^2\mathbf{x}$ define the limits of the space. The w is defined by the following equation.

$$w = 2 \Big/ \left\| {}^2\mathbf{x} - {}^1\mathbf{x} \right\| \tag{4.6}$$

The RBFs can then be set up as a set of equations in the form

$$y = \sum_{i=1}^{N} \omega_i \phi\left(r\right) \tag{4.7}$$

and the weights of coefficients solved for using linear least squares.

The choice of the eight sets of basis functions is fairly arbitrary and dependent

on the user. The polynomials and trigonometric polynomials are included because they are basic basis functions. The linear and spline interpolations are included with the thought that they may better approximate the function in the presence of the inherent raggedness of much of the learning data stored in the action-value function. In essence, they may have a smoothing effect. The RBFs are included because they are a well known and popular set of functions among the neural network community and the function approximation community. It is also possible that data in this research is better suited to approximation by RBFs. Essentially, the wide array of basis function sets allows the GA ample opportunity to find a suitable approximation. Note that it is possible to easily increase or decrease the number of basis function sets by modifying the bit strings of the GA such that the first gene has more or fewer bits, respectively.

2. Fitness Calculation

As stated above, the fitness of a given hypothesis in the GA is a measure of how good it is. The GA in this research is searching for a set of basis functions that best fits the data when using linear least-squares to calculate the weight. The logical fitness choice should therefore be related to how well the true data, $\mathbf{y}$, matches with the approximation of the data, $\tilde{\mathbf{y}}$. There are a number of equations that can be used for this comparison, but for simplicity, the equation used in this GA is the mean squared error between the true and approximated data shown in Eq. 4.8.

$$Fitness\left(h_i\right) = \frac{1}{m}\sum_{j=1}^{m}\left(y_j - \tilde{y}_j\right)^2 \tag{4.8}$$

where m is the number of y-values. Obviously, the GA is trying to minimize the fitness and thus the difference between the true y-values and the approximated $\tilde{y}$-values.

3. Integration in Q-learning Algorithm

As will be discussed in the next chapter, the learning of the action-value function is paused periodically to do an assessment of the current state of the function. This assessment can include either the application of the GA to approximate the action-value function and an analysis of that approximation or a policy comparison method that will be introduced in Chapter V or both.

The Q-learning algorithm stores tabular values the action-value function in tables. There is a table for each action available to the agent. The first several columns contain the state variables, and each row describes the full state. The final column contains the preference or action-value for each state for the action the table represents. These table are what the GA targets to be approximated. Since the table for each state can be markedly different from the others, it was decided by the author that each table of the action-value function should be approximated separately. This means that if there are four action, then there are four tables to represent those actions, and thus there will be four sets of basis functions and associated weights to represent the approximation.

Let the table for the action-value function for $a_1 \in A$ be $Q(s, a_1)$. The approximation of the action-value function for a_1 is

$$\tilde{Q}(s, a_1) = \phi(s, a_1)\, \omega_{\mathbf{a_1}} \tag{4.9}$$

where $\phi(s, a_1)$ and $\omega_{\mathbf{a_1}}$ denote the basis functions and weights, respectively, associated with the approximation of the action-value function table for action a_1. There are similar approximations for all of the tables.

When the learning is paused and the GA used to make an approximation the bit strings representing the basis functions are recorded. The approximated action-

value functions is *not* used in lieu of the tabular action-value function when learning recommences. However, any changes in the bit strings recorded each time the function is approximated is tracked. It is noted when the bit string for each table no longer changes. If the user chooses, this can be used as a stopping criterion for the algorithm, meaning that when the sets of basis functions no longer change, then learning is terminated. The problem that occurs with this choice is that there is not guarantee that the action-value function and policy it represents has converged to a good and usable level. Therefore, another option is to run the policy comparison method, to be discussed in Chapter V, using the approximations instead of the tabular action-value function and use that as the stopping criterion. As will be discussed shortly, this will provide more assurance that the approximated action-value function has converged to an acceptable policy.

C. Illustrative Examples

The use of this GA is illustrated in two simple examples. The purpose of the examples is to show how the GA is applied to a data set and the output thereof. Each example is a simple data set from some simple arbitrary function. The GA is then applied to this data set and determines a bit string encoding the basis function set and degree or number of knots or centers that best approximates the data set. The first example is a 2^{nd} degree polynomial with one independent variable, and the second example is a 4^{th} degree polynomial with two independent variables.

1. Simple Polynomial

This example is a 2^{nd} degree polynomial of the following form:

$$y = 2x^2 + x - 3 \tag{4.10}$$

The data set is generated by taking 101 evenly distributed points over the interval $0 \leq x \leq 10$ and evaluating them using Eq. 4.10. Using the GA parameters listed in Table I, the GA is applied to this data set with "perfect measurements" and this data set with normally distributed noise added.

a. Without "Noise"

For the basic set of data for Eq. 4.10, the GA took 51 generations to sift through the basis function sets to determine the best approximation given the fitness measure of the GA. Figure 7 shows the best and mean fitness for each generation of the population of the GA for this case and the average distance between the members of the population for each generation. The average distance is a measure of the diversity of a population. If the population has high diversity, then its members are quite different from each other and average distance is large. If the population has low diversity, then its members are similar to each other and average distance is small.

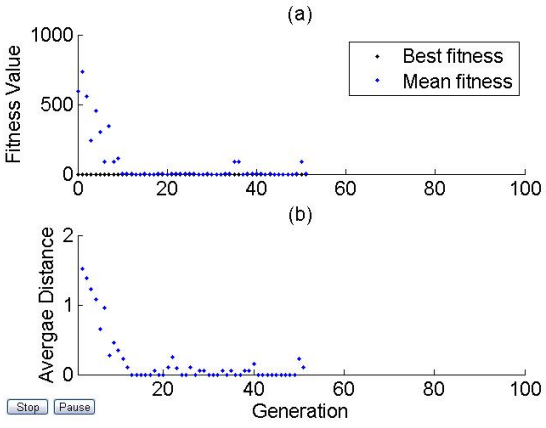

Fig. 7. 2^{nd} Degree Polynomial - (a) Fitness and (b) Average Distance

The mean fitness for the initial population is above 500 for the first several generations. It decreases rapidly and is near 0 within 10 generations. The mean and

best fitness remain near 0 for the remainder of the generations. The reason for the the best fitness never deviating from near 0 and only getting better as the generations progress is that there are elite children that are carried over unchanged from one generation to the next. This ensures that the best can only get better. There are several instances in which the mean fitness jumps away from 0 to about 100. This occurs when one or more bit strings in the new generation has an "unusual" mutation or crossover. There is more variation in the average distance between individuals. The sharply decreasing distance in the first 10 generations reflects the behavior of the mean fitness. The remainder of the generations shows more activity than the mean fitness, however. This activity is due to there being many bit strings in the population that have good fitness but are quite different from each other. This suggests that there are still many possibilities that could produce good approximations of the data set in question.

The final output from the GA for this data set is the bit string

$$0111001 \tag{4.11}$$

which translates to a spline interpolation with 11 knots. The x- and y- values for these 11 knots are listed in Table III. This approximation has a fitness value of 6.48×10^{-29}. True, this is not an exact reproduction of the polynomial, but the fitness value indicates that it is an extremely good approximation. Mathematical error within the GA is likely the reason why this approximation became the final output of the GA rather than the original polynomial.

b. With "Noise"

Noise with a mean of 0 and standard deviation of 0.001 is added to the set of data for Eq. 4.10. The GA for this case searched over the sets of the basis functions for

Table III. Knot Locations for Spline Interpolation Approximation of $y = 2x^2 + x - 3$

x	y
0	-3
1	0
2	7
3	18
4	33
5	52
6	75
7	102
8	133
9	168
10	207

51 generations. Figure 8 shows the best and mean fitness for each generation of the population of the GA for this case and the average distance between the members of the population for each generation.

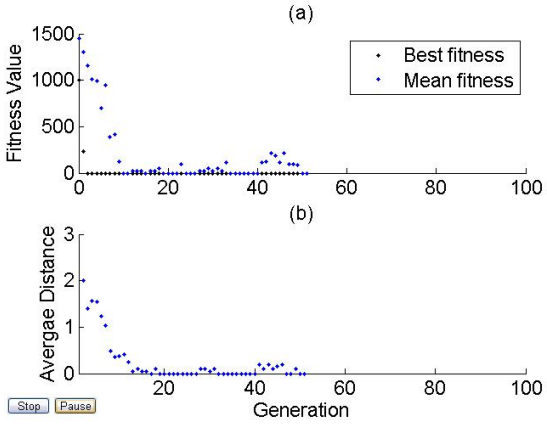

Fig. 8. 2^{nd} Degree Polynomial with Noise - (a) Fitness and (b) Average Distance

The mean fitness for the initial population is much larger for this data set, starting at almost 1500 for the first generations. It decreases rapidly and is near 0 within 15 generations. The mean fitness is more active for this data set than that without noise, but it does remain less than 250 for the remainder of the generations and near 0 for about half of the remainder. The best fitness remains near 0 for the majority of the generations. The average distance again mirrors the activity in the mean fitness. The average distance decreases sharply from a starting value of 2 to near 0 within 15 generations.

The final output from the GA for this data set is the bit string

$$0000010 \tag{4.12}$$

which translates to a 2^{nd} degree polynomial. The approximation determined by the

GA is

$$\begin{aligned} \tilde{y} &= 2.00x^2 + 0.99999999999996x - 3.00043256481146 \\ &= 2x^2 + x - 3 \end{aligned} \tag{4.13}$$

This approximation has a fitness value of 1.34×10^{-26}. In this case, the original polynomial is reproduced by the GA to within mathematical precision.

Figure 9 illustrates the true data with and without noise and their approximations. As is readily apparent, there is no discernible visual difference between the true data and the approximation. Indeed, this is to be expected since the fitness values for both approximations are essentially 0, so there would only be concern if there were a visual difference when the mathematics indicate that there should not be any.

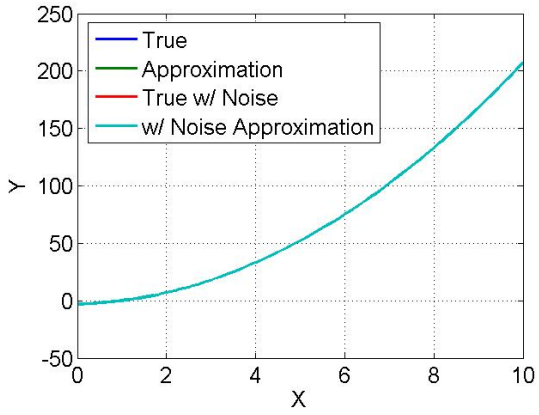

Fig. 9. Comparison of the True and Approximated Data for $y = 2x^2 + x - 3$

2. Rosenbrock's Function

This example Rosenbrock's function, which is of the following form:

$$\begin{aligned} f(x_1, x_2) &= 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 \\ &= 100x_1^4 - 200x_1^2x_2 + 100x_2 + x_1^2 - 2x_1 + 1 \end{aligned} \tag{4.14}$$

The data set is generated by taking 1681 evenly distributed points over the space $-2 \leq x_1 \leq 2$ and $-2 \leq x_2 \leq 2$ and evaluating them using Eq. 4.14. Using the GA parameters listed in Table I, the GA is applied to this data set as is and this data set with normally distributed noise added.

a. Without "Noise"

For the set of data for Eq. 4.14, the GA again took 51 generations to sift through the basis function sets to determine the best approximation. Figure 10 shows the best and mean fitness for each generation of the population of the GA for this case and the average distance between the members of the population for each generation.

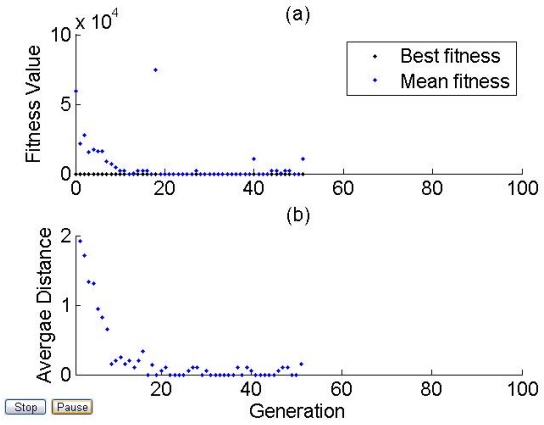

Fig. 10. Rosenbrock's Function - (a) Fitness and (b) Average Distance

The general trend of the mean fitness, best fitness, and average distance is the same as for the previous example. There is a sharp decrease in the mean fitness and average distance from 60000 and 2, respectively, to near 0 within 20 episodes. Both take longer to approach 0 than before because there are more data points to fit the approximation to. This leads to an increase in error between the true and approximated values and a subsequent increase in fitness. This phenomenon is also evident in the average distance. There is more activity in the later episodes than seen before. The reason is that there are fewer approximations that give what could be considered a "good" approximation since there is more data to fit. However, that does not mean that there are no good approximations. The best fitness is close to 0 for the entirety of the GA's search. This indicates that there is at least one child in each generation that has a small fitness value.

The final output from the GA for this data set is the bit string

$$0000100 \tag{4.15}$$

which translates to a 4^{th} degree polynomial. The approximation determined by the GA is

$$\begin{aligned} f(x_1, x_2) &= 100x_1^4 - 200x_1^2x_2 + 99.\bar{9}x_2 + 0.\bar{9}x_1^2 - 1.\bar{9}x_1 + 1.00 \\ &= 100x_1^4 - 200x_2x_1^2 + 100x_2 + x_1^2 - 2x_1 + 1 \\ &= 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 \end{aligned} \tag{4.16}$$

where the weights for the missing pieces of the polynomial in Eq. 4.16 are 0. This approximation is a near exact reproduction of Rosenbrock's function and has a fitness value of 2.23×10^{-23}.

b. With "Noise"

Noise with a mean of 0 and standard deviation of 0.001 is added to the set of data for Eq. 4.14. The GA once again took 51 generations to sift through the basis function sets to determine the best approximation. Figure 11 shows the best and mean fitness for each generation of the population of the GA for this case and the average distance between the members of the population for each generation.

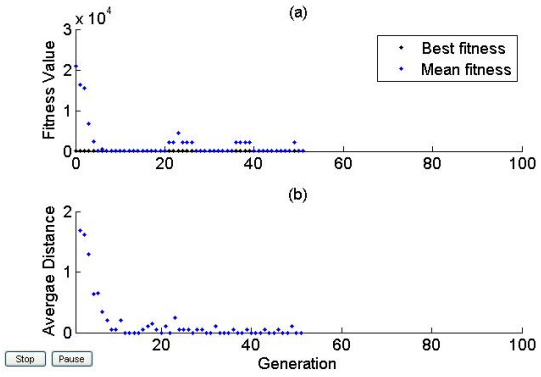

Fig. 11. Rosenbrock's Function with Noise - (a) Fitness and (b) Average Distance

The general trend of the mean fitness, best fitness, and average distance is the same as for data without noise. There is a sharp decrease in the mean fitness and average distance from 20000 and 2, respectively, to near 0 within 10 episodes. Interestingly, both tend toward 0 more quickly than the data without noise, though there is more activity evident in the average distance for the remainder of the generations. The best fitness is again close to 0 for the entirety of the GA's search, though it is on the order of 1×10^{-7} rather than 1×10^{-23}.

The final output from the GA for this data set is the bit string

$$0001111 \tag{4.17}$$

which translates to a 15^{th} degree polynomial. The approximation determined by the GA is

$$\begin{aligned} f(x_1, x_2) &= 100.0017x_1^4 - 200.0009x_1^2x_2 + 100.0021x_2 + & (4.18) \\ &+ 0.9993x_1^2 - 2.0008x_1 + 0.9998 & (4.19) \\ &= 100x_1^4 - 200x_2x_1^2 + 100x_2 + x_1^2 - 2x_1 + 1 \\ &= 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 \end{aligned}$$

where the weights for the missing pieces of the polynomial in Eq. 4.18 are $|\omega_i| < 0.014$. This approximation is a near exact reproduction of Rosenbrock's function and has a fitness value of 9.03×10^{-7}.

Figure 12 illustrates the true data with and without noise and their approximations. There is again no discernible visual difference between the true data and the approximation. This shows that the GA is able to determine a set of basis functions that adequately approximates the data. The presence of noise leads to approximations that are less accurate in terms of fitness and thus mean square error, but that error is still so small that it does not present a problem.

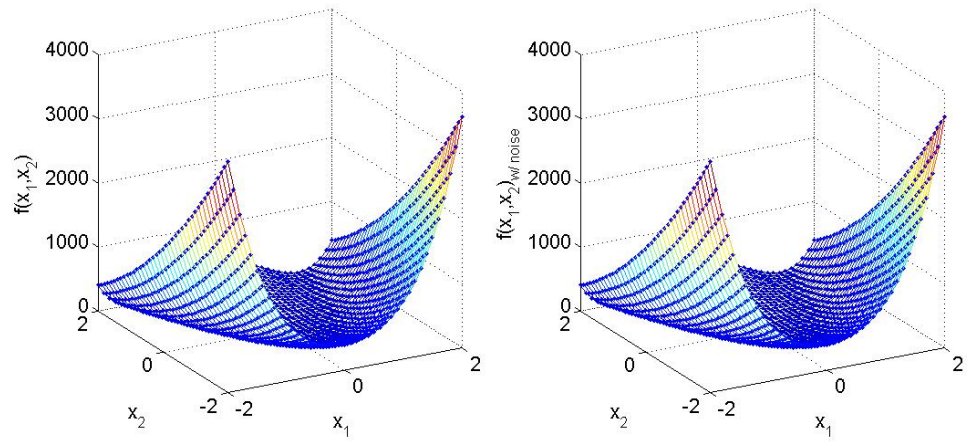

Fig. 12. Comparison of the True and Approximated Data for Rosenbrock's Function

CHAPTER V

POLICY COMPARISON (PC)

A. Problem Definition

The multi-resolution discretization method provides a means of learning the action-value function, $Q^{\pi}(s,a)$, for a fixed policy, π, in progressively finer detail. Rather than blindly allowing the agent to learn for the entire number of user defined episodes, stopping criteria based on the learned policy are introduced.

In Q-learning all of the information is stored in the form of the action-value matrix, often in the form of a table. The learned greedy policy itself is not represented explicitly or with any sort of model. However, the policy and associated value function can be easily extracted from the action-value function in a few simple steps. Recall the relationship between a value function and an action-value function introduced in Chapter II Eq. 2.14. A similar relationship exists for the greedy policy, namely

$$\pi(s) = \arg\max_{a} Q(s,a) \tag{5.1}$$

where $\pi(s)$ is the action associated with the maximum preference over the set of actions for the state. In terms of the approximated action-value function determined by the genetic algorithm that was introduced in the previous chapter, the same relationship holds for the approximated policy:

$$\tilde{\pi}(s) = \arg\max_{a} \tilde{Q}(s,a). \tag{5.2}$$

As a side note, a representation of the value function can then be easily calculated:

$$V(s) = \max_{a} Q(s,a). \tag{5.3}$$

for the tabular action-value function and

$$\tilde{V}(s) = \max_{a} \tilde{Q}(s, a). \tag{5.4}$$

for the approximated action-value function. This relationship will be used for visual analysis in later chapters.

Two stopping criteria are added to the Q-learning algorithm and form the third and final addition that makes up the new overall algorithm developed here. These two criteria are a direct policy comparison and a performance based policy comparison, which are periodically applied to the learned action-value function to determine if the action-value function has converged to a usable data set. Both criteria use the relationships in Eq. 5.1 and 5.2 for the tabular and approximated action-value function, respectively. Both of the stopping criteria introduced in this chapter must be met for learning at the current level of discretization to be terminated, otherwise learning continues. These criteria are described in the following sections.

B. Policy Comparison vs. Policy Iteration

The direct policy comparison stopping criteria is a simple and expedient way to track the change in the policy extracted from an action-value function as that function evolves during learning. This policy comparison is carried out in a short series of steps:

1. Pause learning after n episodes
2. Extract current greedy policy, $\pi_i(s)$, from action-value function, where i is the number of elapsed episodes divided by n
3. Directly compare $\pi_i(s)$ and $\pi_{i-1}(s)$

Table IV. Policy Comparison Example - Part 1

State	$\pi_1(s)$	$\pi_2(s)$
s_1	3	3
s_2	1	3
s_3	2	1
s_4	2	2
s_5	3	1
s_6	3	2
s_7	2	2
s_8	1	1
s_9	3	3
s_{10}	2	2

4. If change in policy, $\Delta\pi$, is $< \varepsilon$, then stopping criteria #1 is achieved.

Here ε is a small number and is usually set as 5% for this research. This same series of steps holds for the approximated action-value function. Also, after learning is initialized and the first set of n episodes elapse, the policy, $\pi_1(s)$, is extracted and stored only as there is no $\pi_0(s)$. This comparison method is fully utilized starting after the second set of n elapsed episodes.

To better understand this methodology, consider a simple example. A problem is learned and has 10 states and 3 actions, where $(s_1, s_2, \ldots, s_{10}) \in S$ and $(a_1, a_2, a_3) \in A$, respectively. The action-value function associated with this problem is not important. Simply consider the extracted policy after n and $2n$ episodes shown in Table IV. There is agreement between the two policies for 6 out of the 10 states, so $\Delta\pi = 0.4$ or 4%. After another n episodes, the policy is again extracted

Table V. Policy Comparison Example - Part 2

State	$\pi_2(s)$	$\pi_3(s)$
s_1	3	3
s_2	3	3
s_3	1	1
s_4	2	2
s_5	1	1
s_6	2	2
s_7	2	2
s_8	1	1
s_9	3	3
s_{10}	2	2

and compared. This new policy is shown in Table V. Now there is perfect agreement between the two policies, so $\Delta\pi = 0$ or 0%. If ε was set as 5%, or even 0%, then this problem has now satisfied the first stopping criteron.

It is entirely possible that a lack of change in the extracted policy is not an indication that the action-value function has converged to a usable function. It could simply be a momentary aberration and would begin to change again if another n episodes is allowed. To prevent this from occurring a second stopping criterion is introduced. First, however, I'll briefly describe policy iteration, a well known and basic learning method, to show that the simple method introduced here is quite different and to avoid any future confusion.

Policy iteration is defined by Sutton and Barto as the process of finding an optimal policy by using policy π that has been improved using V^π to produce a better

policy, π'. $V^{\pi'}$ can then be computed and improved to produce a better policy, π''.[81] This generates a sequence of monotonically improving policies and value functions:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} V^*. \tag{5.5}$$

This denotes a series of policy *evaluations*, $\xrightarrow{E}$, followed by policy *improvements*, $\xrightarrow{I}$. The policy evaluation step calculates the value, $V_i = V^{\pi i}$, of each state given a policy π_i if π_i were to be executed (see Eq. 5.6).

$$V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left(\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right) \tag{5.6}$$

The policy improvement step calculates a new policy π_{i+1} using a one-step look-ahead based on V_i as in Eq. 5.7.[81, 179]

$$V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left(\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right) \tag{5.7}$$

A finite MDP has only a finite number of policies, so this process must converge to an optimal policy and optimal value function in a finite number iterations. The policy iteration algorithm terminates when the there is no further change.[179] A modification to the policy iteration algorithm called asynchronous policy iteration picks a subset of states and applies either policy improvement or value iteration rather than updating the policy for all states at once. Policy iteration does have a drawback in that each time the algorithm calls for policy evaluation, it could require an iterative computation requiring multiple sweeps through the state-space, which adds complexity to the process.

The difference between this and the policy comparison method described above is that policy iteration continually updates and uses the policy and value function it is learning. The policy comparison method is strictly that, a comparison of the current greedy policy to the previous greedy policy. Since Q-learning is an off-policy

method, it can use any preset policy as it learns the greedy policy as discussed in Chapter II. Hence, the Q-learning continues to use that predefined policy regardless of the state to action mapping defined by the most recently extracted greedy policy.

C. Performance Based Policy Comparison - Monte Carlo Simulation

The second stopping criterion implemented is based on the performance of the current policy. During the pause in learning after every n episodes, the policy comparison is conducted and then this performance based policy comparison. The performance based policy comparison measures performance by conducting a set of Monte Carlo simulations. It is referred to as Monte Carlo in the sense that initial conditions are taken from a uniform distribution and a large number of simulations are conducted and recorded. In each simulation the agent is initialized in a random non-goal state within the region of the current level of discretization. It then uses the current learned greedy policy, meaning it exploits its current knowledge of the state-space, to navigate through the state-space to find the goal. A *success* occurs when the agent navigates from the random initial state to a goal state without encountering a boundary. A *failure* occurs when the agent either encounters the outer most boundary of the state-space, the boundary of the current level of discretization, or gets "lost" and wanders around the state-space. This simulation is conducted a predefined number of times, usually 500 simulations in this research, and each success is recorded. The success percentage is then calculated using Eq. 5.8.

$$\% \text{ Success} = \frac{\# \text{ of Successes}}{\text{Total } \# \text{ of Simulations}} \tag{5.8}$$

When this success percentage is above some threshold, usually 98% in this research, the second stopping criterion is satisfied. Both the first and the second stopping

criterion must be met for the learning at the current level of discretization to be terminated and learning continued at a finer level of discretization as necessary.

CHAPTER VI

MULTI-RESOLUTION STATE-SPACE DISCRETIZATION WITH A GENETIC ALGORITHM FOR FUNCTION APPROXIMATION AND POLICY COMPARISON PERFORMANCE ANALYSIS (MGAP)

There are several features that make up this new modified version of Q-learning. To help illustrate both the setup and the novel aspects, consider the flowcharts in Figures 13, 14, and 15. These flowcharts represent the basic Q-learning algorithm, the LSPI algorithm, which has several similar components as compared to Q-learning with MGAP, and Q-learning with MGAP, respectively.

The Q-learning algorithm was described in Chapter II, but the flowchart shows the episodic nature of the algorithm described by the pseudocode in that chapter. Learning in the form of updating the action value function (Eq. 2.16) continues until some form of stopping criterion is reached, namely the user selected total number of episodes. To reiterate, the output of Q-learning is the action-value function, $Q(s, a)$, that enumerates the preference or value of an action, a, for every state, s. For Q-learning on a discrete state-space, the action-value function can be used as a (often) large table lookup to choose the action with the highest preference for a given state. However, it is not always feasible to carry large tables in many applications, such as an autonomous reconfigurable MAV, which has limited computational capacity. Therefore, post-processing of the data often includes approximating the action-value function using some form of function approximation (see Eq. 6.7), e.g. K-Nearest Neighbors (KNN), Sequential Function Approximation (SFA)[173, 174], GLO-MAP, least-squares, *etc.* The final output shown in Figure 13 is thus the approximated action-value function, $\tilde{Q}(s, a)$.

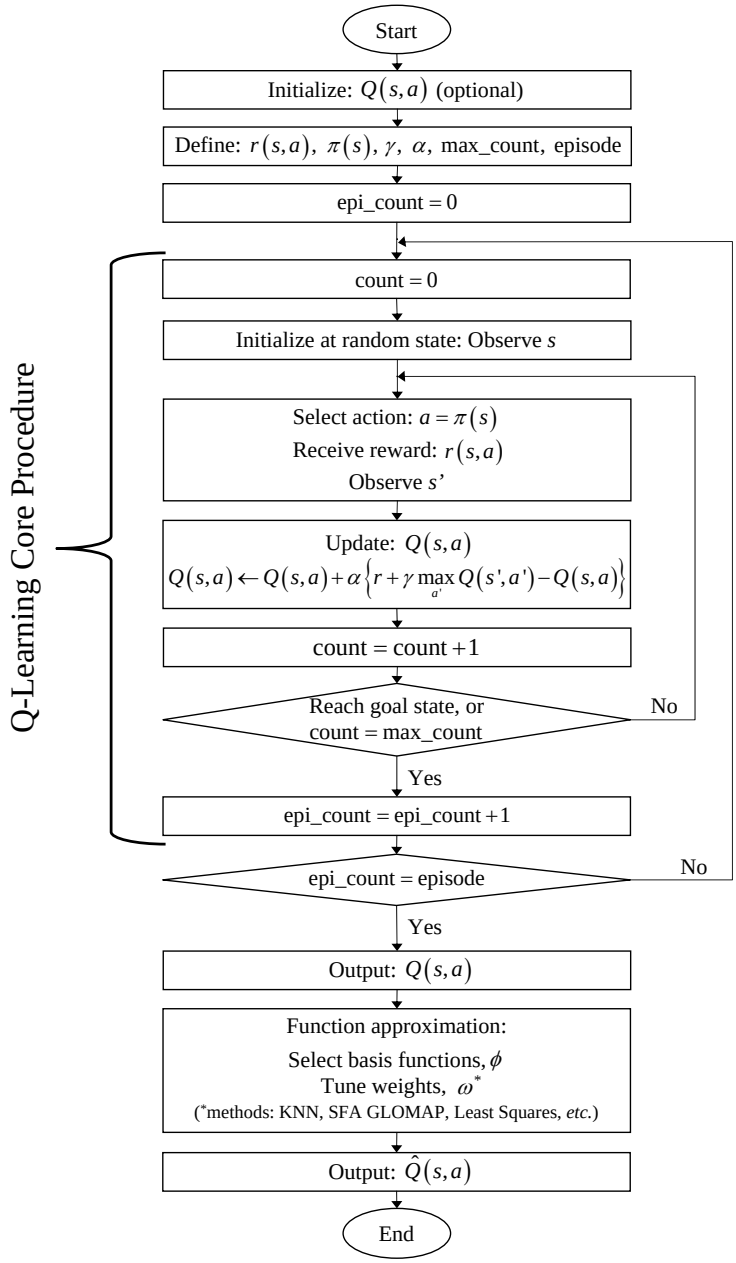

Fig. 13. Flowchart of Q-Learning

A. Full Algorithm Description

The flowchart in Figure 14 represents the Q-learning with MGAP algorithm. The four core elements are the basic Q-learning algorithm (described in Chapter II), the multi-resolution state-space discretization method (described in Chapter III), the genetic algorithm for function approximation (described in Chapter IV), and the policy comparison and performance measure (described in Chapter V). Succinctly, Q-learning with MGAP uses the basic Q-learning algorithm to learn the action-value function of the given problem for an initial state-space discretization. After a preset number of episodes, such as after 50 episodes, the learning is paused. At this point the genetic algorithm is launched to approximate the current action-value function. Using the approximation of the form

$$\tilde{Q}\left(s, a_i\right) = \phi\left(s, a_i\right) \omega_{\mathbf{a_i}}, \tag{6.1}$$

the policy is extracted using the relationship in Eq. 6.2.

$$\tilde{\pi}_i\left(s\right) = \arg\max_a \tilde{Q}\left(s, a\right) \tag{6.2}$$

Also, at this juncture, a performance test is conducted. This performance test consists of a set of simulations in which the agent starts at a random initial state and is told to exploit its knowledge, i.e. use a greedy policy, to find the goal. A percentage of success is recorded. These steps are repeated periodically until the change in policy is small, $\Delta\pi$, is $< \varepsilon$, and the percentage of performance success is high. Once the action-value function has converged for this level of discretization, the state-space around the region(s) of interest is rediscretized to a finer degree according to Eq. 6.3 and learning commences in this smaller area.

$$r_{(fine)}\left(s, a_{(fine)}\right) = r_{(fine+1)}\left(s, a_{(fine+1)}\right) \tag{6.3}$$

This process continues until the preset finest level of discretization is reached. The final output is an approximation of the action-value function that learned the area of interest in fine detail and that is empirically shown to perform the task well via the policy comparison and policy performance measure.

B. Summary of Least-Squares Policy Iteration

As stated in Chapter I, LSPI is a least-squares method for learning an action-value function based on least-squares approximation theory. The flowchart in Figure 15 is based on the derivation and pseudocode of the algorithm provided in Ref. [140]. This algorithm conducts learning by generating a set of samples, or tuples, of the form (s, a, r, s') for the problem at hand and incorporating a set of user selected basis functions stored in ϕ. Note that the basis functions are selected *a priori*. The policy, $\pi(s)$, a state to action mapping, is initialized (often as all zeros) at the beginning. During the initial iteration through the algorithm, the update equations for the A matrix and b vector are updated according to the update Eqs. 6.4 and 6.5 using the previously generated tuples.

$$\tilde{\mathbf{A}} \leftarrow \tilde{\mathbf{A}} + \varphi(s,a)\left(\varphi(s,a) - \gamma\varphi(s', \pi(s))\right)^T \tag{6.4}$$

$$\tilde{b} \leftarrow \tilde{b} + \varphi(s,a)\, r \tag{6.5}$$

When the set of tuples is exhausted, the weights for the basis functions are calculated by Eq. 6.6.

$$\begin{aligned} \tilde{b} &= \tilde{\mathbf{A}}\omega' \\ \omega' &\leftarrow \tilde{\mathbf{A}}^{-1}\tilde{b} \end{aligned} \tag{6.6}$$

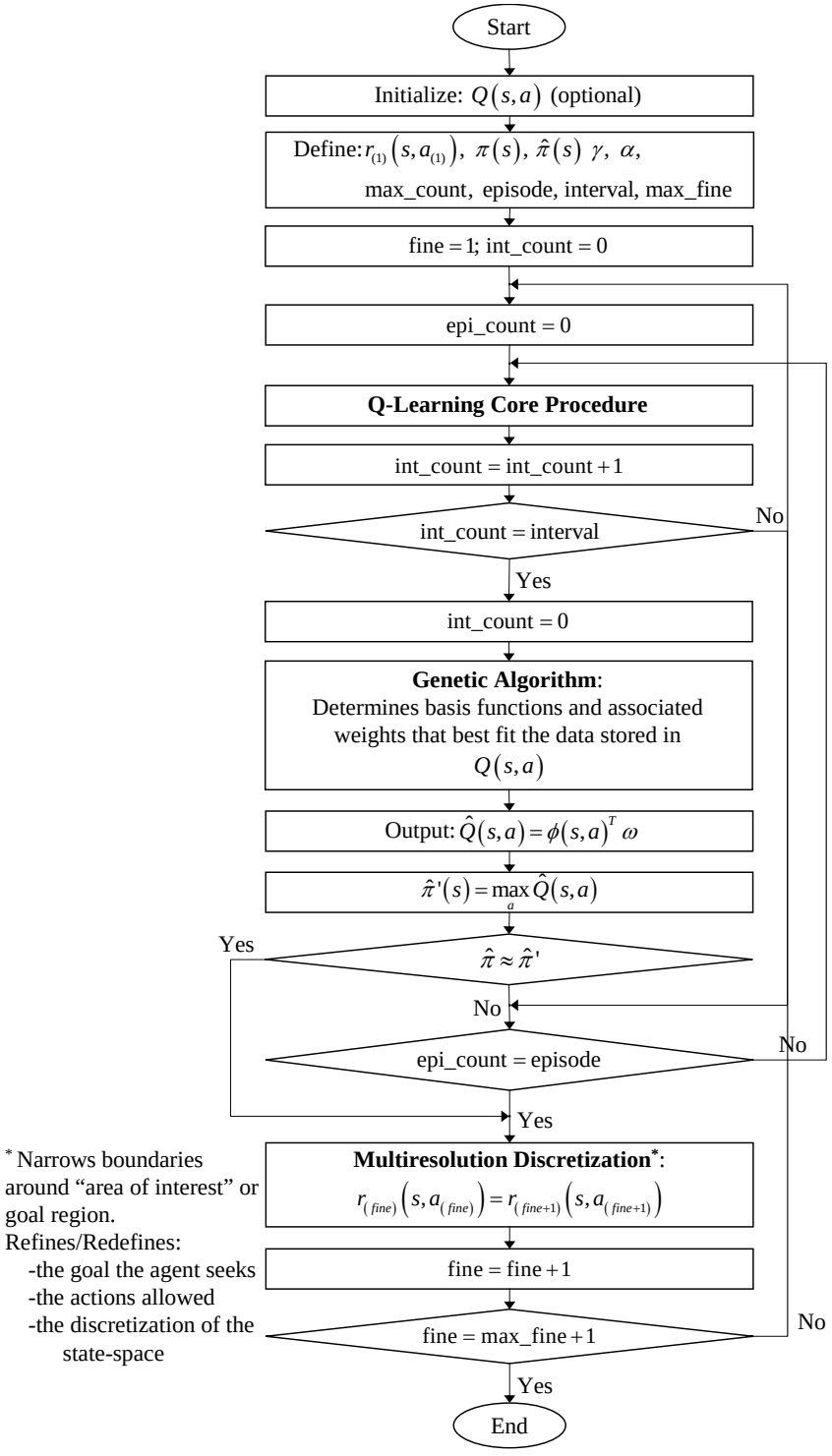

Fig. 14. Flowchart of Proposed Algorithm

This yields the initial approximation of the action-value function of the form:

$$\tilde{Q}(s,a) = \phi(s,a)^T \omega \tag{6.7}$$

From this approximation, the policy can be extracted. These steps repeat (update A and b, calculate weights, extract policy) until the change in the policy from one iteration to the next is very small. Lagoudakis and Parr report that, for the problems they applied the algorithm to, only a small number of iterations are necessary.[140] The authors also report that the selection of basis functions is important and can affect the convergence of A, b, and, subsequently, $\pi(s)$. Basically, the basis functions are selected first and the user hopes that they yield a good solution. If not, new functions are selected, and the algorithm is rerun.

C. Comparison of MGAP vs. LSPI

The greatest benefit of LSPI, in the opinion of this author, is that all that needs to be stored for implementation after learning is $\phi(s,a)$ and ω, each of which are $k \times 1$, where k is the number of basis functions multiplied by the number of actions. Also, rather than updating a possibly large tabular action-value function with the tuples, only A and b need updating and are $k \times k$ and $k \times 1$, respectively, regardless of the number of tuples.

By comparison, Q-learning with MGAP autonomously selects and tests many sets of basis functions online to determine what best approximates the action-value function, thus avoiding this often time consuming process. In addition, the derivation of LSPI limits the function approximation to a global least-squares approximation dependent on the pre-selected basis functions. To use a more sophisticated function approximation technique would require that the entire algorithm be re-derived. The

D – Source of samples (s, a, r, s')
k – Number of basis functions
ϕ – Basis functions
γ – Discount factor
ε – Stopping criterion
π_0 – Initial policy, given as ω_0 (default: $\omega_0 = 0$)

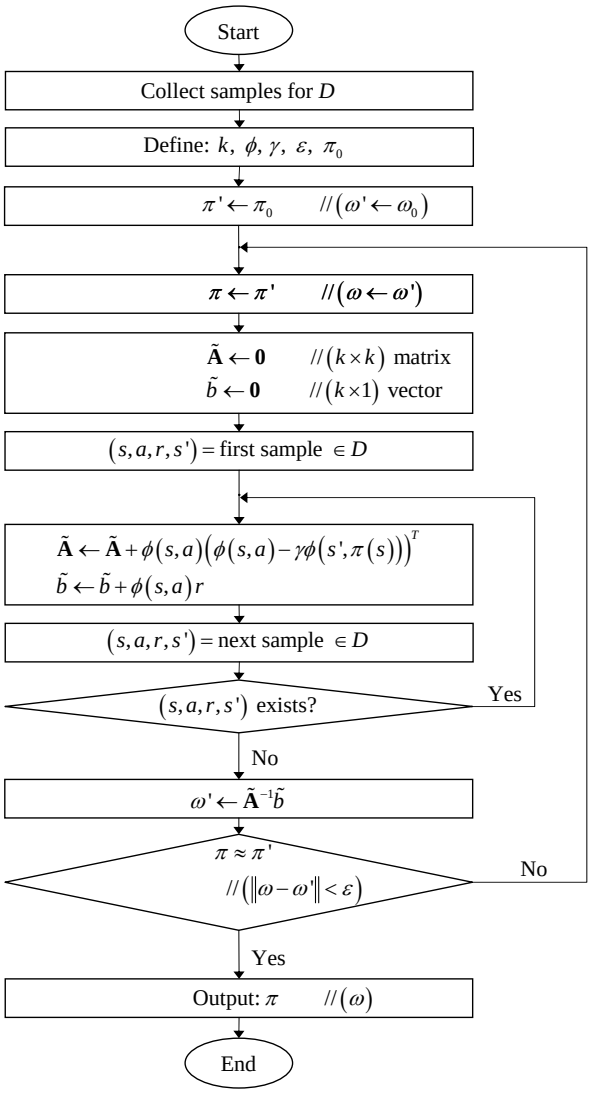

Fig. 15. Flowchart of LSPI

modular format of MGAP, however, allows for the easy integration of approximation techniques. Currently, the GA employs a global least-squares approximation as well, but replacing the approximation with a more sophisticated method, such as GLOMAP, is a simple matter of communication between Q-learning and the enhancements.

Unlike the policy iteration of LSPI, the policy comparison method not only determines when the policy converges, it also determines when the policy has converged to a policy that can successfully guide the agent to the goal for both the tabulated and the approximated action-value function. Finally, the multi-resolution state-space discretization enhancement guides the agent to focus on Regions Of Interest where additional and more detailed learning is determined necessary. Due to the global approximation nature of LSPI and similar learning algorithms, focusing learning in specific areas will not necessarily be reflected in the final output of the algorithm. The approximation may not reflect the details learned in these regions. By discretizing the state-space, the detailed learning in specific regions is more readily reflected and the approximation can be tailored accordingly.

CHAPTER VII

BENCHMARK DYNAMIC SYSTEM EXAMPLE - INVERTED PENDULUM

The inverted pendulum is a classic system on which to test new control algorithms. It is a simple unstable system that requires a certain amount of finesse to control. The simple inverted pendulum system shown in Figure 16 has four degrees of freedom: cart position, x, cart velocity, $\dot{x}$, pole angular position, θ, and pole angular velocity, $\dot{\theta}$. This system has one equilibrium point, which is unstable, at $\theta = 0\,\text{deg}$.

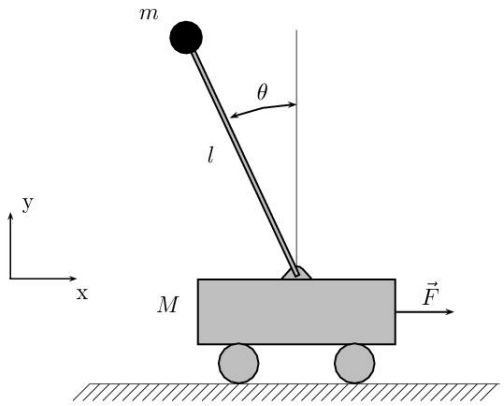

Fig. 16. Inverted Pendulum System

As stated above, this simple system, and derivatives thereof, is often used to test new controllers. For example, the system in which two pendulums are to be balanced was addressed using a double CMAC network.[180] The authors effectively learned to balance the system by first splitting it up into two single pendulum problems and then combining them with some extra learning to balance the double system. There are also many who have controlled this system with fuzzy logic controllers in one form or another. One such reference used the Takagi-Sugeno model to do the full swing

up and balance maneuver of the single inverted pendulum with friction.[181] Another used a fuzzy controller to do the swing up maneuver and the balancing after a certain threshold using a linear quadratic regulator.

Q-learning has been used a number of times as well. A hierarchical reinforcement learning controller for the double inverted pendulum was designed by Zheng, *et al.*, that uses a similar setup as the CMAC method, but with standard Q-learning as the algorithm of choice. The system is again split into two single pendulum problems and combined into the double pendulum.[182]

There are many more references regarding control via linear quadratic regulators, fault tolerant adaptive control, all manner of reinforcement learning methods, modern control, and classical control. In general, this type of system is a benchmark control problem on which controllers ought to be tested. It is versatile in the sense that it can be made more difficult by using the full nonlinear equations of motion and/or by adding friction or damping effects. It can be made simpler by linearizing about the equilibrium point and/or negating damping effects.

A. Inverted Pendulum Model

The inverted pendulum is simulated as a simple, nonlinear dynamic system. Friction is assumed negligible. The derived equations of motion for this system are

$$\begin{aligned} \ddot{x} &= \frac{-mg\cos\theta\sin\theta - m\dot{\theta}^2 l\sin\theta + F}{M+m-m\cos^2\theta} \\ \ddot{\theta} &= \frac{F-(M+m)g\tan\theta + m\dot{\theta}^2 l\sin\theta}{-\left(\frac{Ml+ml}{\cos\theta}\right)+ml\cos\theta} \end{aligned} \tag{7.1}$$

The constants for the system are listed in Table VI.

The Q-learning controller determines the sign and magnitude of the force to be applied to the system in intervals of $\Delta t = 0.05\,\text{sec}$. In essence the controllers act as

Table VI. Constants of the Inverted Pendulum System

Parameter	Value
M (kg)	5
m (kg)	1
l (m)	5
g ($\frac{m}{\text{sec}^2}$)	9.81

Sampled Data Regulators as they attempt to balance the pole around the equilibrium point $\theta = 0\,\text{deg}$.

B. Inverted Pendulum Cast as a Reinforcement Learning Problem

Casting a dynamic system as a reinforcement learning problem takes careful consideration. The degrees-of-freedom chosen to be the state variables, such that $s \in S$, to be used by the learning algorithm must adequately capture the dynamics of the system and the goal to be achieved. The dynamic system, and subsequently the state variables, constitute the *environment* with which the agent interacts. This system has four possible degrees-of-freedom, the interdependence of which are captured in the two equations of motion. If one were to rush into casting the system as a reinforcement learning problem, one would include all degrees-of-freedom as state variables, $\left(x, \dot{x}, \theta, \dot{\theta}\right) \in S$. This would be necessary if the goal was to balance the pole around $\theta = 0\,\text{deg}$ *and* restrict the cart to a finite track. Since this is not case for this formulation of the problem and only balancing the pole around $\theta = 0\,\text{deg}$ is of interest with no restrictions on the cart, only the angle and angular velocity of the pendulum mass are considered to be state variables, $\left(\theta, \dot{\theta}\right) \in S$. Note that both equations of motion must be retained and simulated as they are interdependent and the state variables of

Table VII. Inverted Pendulum Q-Learning Example Reward Structure

Bounds	Reward
$\lvert\theta\rvert \leq 2\,\text{deg}$	1
$2\,\text{deg} < \lvert\theta\rvert \leq 12\,\text{deg}$	0
$\lvert\theta\rvert > 12\,\text{deg}$	-1

interest show up in both equations.

The only input into the system is a horizontal force applied to the cart. Thus, the *action* available to the agent is the force to be applied to the system, i.e. $(+F, -F) \in A$. The forces are set at the user's discretion. The action space can be restricted to 2 actions only or have 10 or more. The main effect of action selection is on the complexity of the problem since the more actions available to the agent, the more trials necessary to learn which actions are appropriate for the given state.

The *reward*, r, received by the agent from the environment is based on the current state of the system. Since this is a simple system, the reward structure is kept simple (see Table VII). The reward is set up such that if the agent manages to propel the pendulum near the goal, it receives a positive reward. Additional positive rewards are received if the pendulum is maintained near the goal for multiple time steps. This encourages the agent to learn to balance the pendulum for as many time steps as possible. A negative reward is received if the pendulum falls beyond some angle, and no reward or a neutral reward is received if the angle of the pendulum lies between the two bounds.

For this problem the state-space is *quantized* rather than discretized. Quantized means that the dynamic system is simulated in continuous time and at predefined intervals the state is noted, and the update to the action-value function determined

Table VIII. Inverted Pendulum Q-Learning Parameters

Parameter	Value
Episodes	5000
Δt	0.05 sec
t_f	50 sec
α	0.01
γ	0.7

by the learning algorithm is applied to the nearest learning storage state in the action-value function.

C. Numerical Results

The agent is given the task of learning to balance an unstable system for as many time steps as possible. By setting up the system as a reinforcement learning problem described above, the agent is set to learn how to control the system. Additional parameters for the learning problem are listed in Table VIII. The agent is allowed 5000 episodes over which to learn the controller for the current goal. The initial state for each episode is random and within the neutral or positive bounds of the state-space. Each episode is simulated for a total of 50 sec, and every 0.05 sec the state is noted and the action-value function updated. If the pendulum falls beyond the negative bounds, the episode is terminated and a new episode begun.

For this example, the agent learns using only two algorithms. The agent first uses Q-learning to establish a baseline controller. In the second part, the agent uses Q-learning with AAG in an attempt to learn a better controller. The two controllers are evaluated in a number of ways. First, it is helpful to consider a value function

Table IX. Inverted Pendulum Q-Learning Simulation Initial Conditions

State	Value
x	$0m$
$\dot{x}$	$0m/\sec$
θ	$0.5\deg$
$\dot{\theta}$	$0\deg/\sec$

approximated from the learned action-value function. The value function can show how the learning differs when learning parameters or discretization is modified. The policy can be extracted from the action-value function as described in Chapter V using Eq. 5.1. This equation represents the action associated with the maximum preference or action-value for each state. An approximation of the value function based on both the action-value function and the extracted policy can then be found with Eq. 5.3 from Chapter V.

Since this is a dynamic system, it is also informative to simulate the system with the learned controller in action. Both controllers are evaluated for simulations of 10 sec and 300 sec. The initial conditions for the simulations are listed in Table IX. Control is applied in the same way that it is learned. Every $\Delta t = 0.05$ sec an action or force is selected based on the highest preference in the action-value function given the current state, see Eq. 5.1. That force is applied for 0.05 sec, and then a new action is selected. The 10 sec simulation is intended to show an up close view of the workings of the controller, whereas the 300 sec simulation is intended to prove that controller can balance the pendulum for an extended period of time, in this case for 6000 time steps.

Table X. Inverted Pendulum Policy Color Scheme

Action	Color
$-F$	Blue
F	Red

1. Case 1: Q-Learning

The controller learned using the Q-learning algorithm is restricted to only a couple possible actions. These action are the following forces

$$(-15N, 15N) \in A. \tag{7.2}$$

The reward structure for this case is that listed above in Table VII. The states were quantized to every 1 deg and 1 deg / sec for angular position and velocity, respectively.

The value function and policy for this case are shown in Figure 17. The actions for the policy are color coded and are listed in Table X The value function shows a marked peak crest in the goal region where the agent received positive rewards. This encourages the agent to attempt to remain in this area. There is a sharp drop off in value beyond the goal region. There is also negative regions near the outer bounds of the state-space. The visual representation of the policy shows that the agent probably needs more learning time. One would expect more continuous regions of a given action. The figure and intuition of the system suggests that for a fully converged policy, the upper right half of the state-space should be red and the lower left half blue. Due to the dynamic nature of the system, however, there is no coordinated way to have the agent focus on these regions aside from initializing the many episodes in these regions. For more complex problems, there is no guarantee that there would be

large regions in which one particular action is ideal. Therefore, artificially forcing the agent to initialize in these regions for this simple problem is counterproductive since the method is unlikely to scale up to a more complex problem.

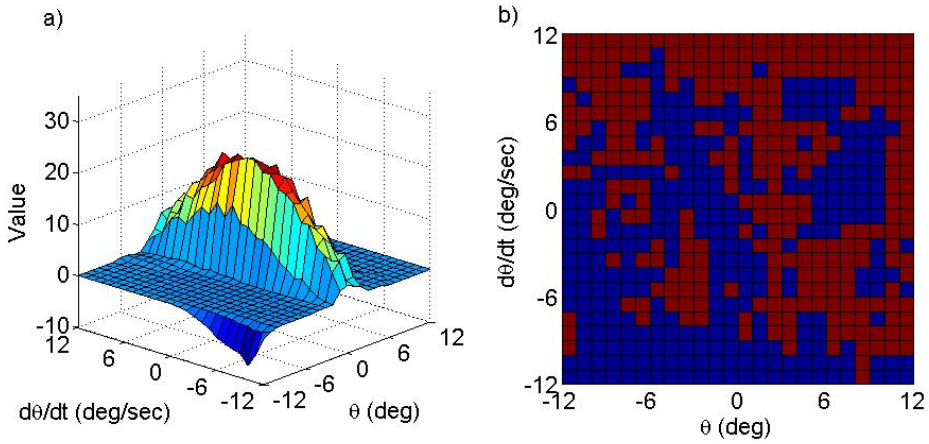

Fig. 17. Basic Inverted Pendulum Controller: Value Function (a) and Policy Representation (b)

Either way, the true test for this kind of learned controller is to actually test the controller. Figures 18 and 19 show the time history and phase diagram for the 10 sec and 300 sec simulations, respectively. The time history in Figure 18 illustrates the jagged nature of this controller resulting from the restricted number of control inputs. Despite this the learned controller balances the pendulum within the goal region of $|\theta| \leq 2\,\text{deg}$ as evidenced by both the time history and the phase diagram. The angular velocity does spike close to $8\,\text{deg}\,/\,\text{sec}$ about 9 sec into the simulation, but the controller is able to recover the system and maintain balance.

The time history in Figure 19 shows a similar response as the controller continues to balance the pendulum. The pendulum is balanced for the full 300 seconds and appears to be stable, but does not get much closer than the oscillation between about $\theta = -1\,\text{deg}$ and $\theta = 2\,\text{deg}$ with the angular velocity approaching $8\,\text{deg}\,/\,\text{sec}$

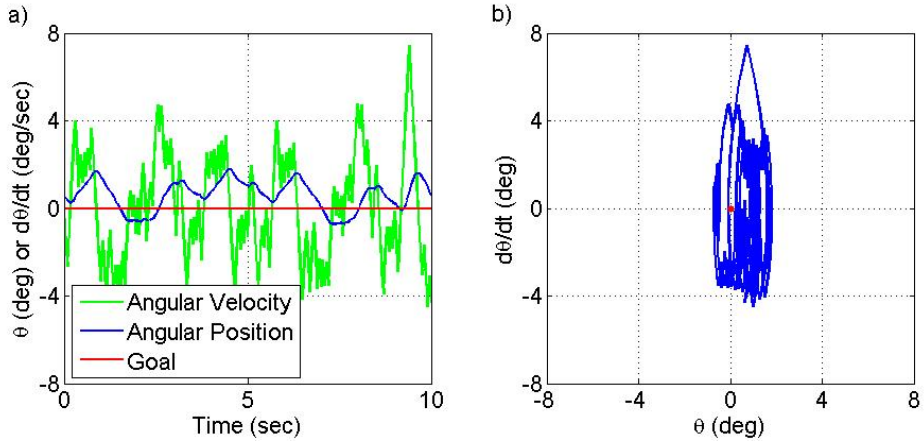

Fig. 18. Basic Inverted Pendulum Controller, 10 Seconds: Time History (a) and Phase Diagram (b)

Table XI. Inverted Pendulum Action Sets and State Quantizations

Discretization Level	Action Set	State Quantization $\left(\theta, \dot{\theta}\right)$
1	$(-15N, 15N) \in A$	$(1\,\text{deg}, 1\,\text{deg / sec})$
2	$(-5N, 5N) \in A$	$(0.2\,\text{deg}, 0.2\,\text{deg / sec})$
3	$(-1N, 1N) \in A$	$(0.04\,\text{deg}, 0.04\,\text{deg / sec})$

many times. The agent achieved its goal of learning a controller that balances around $\theta = 0\,\text{deg}$ and within $|\theta| \leq 2\,\text{deg}$, but the response leaves much room for improvement.

2. Case 2: Q-Learning with AAG

The controller learned using Q-learning with AAG is a little more complicated. There are three levels of discretization or quantization on which the agent is allowed 5000 episodes for learning in each. The action sets and state quantizations for each level are listed in Table XI.

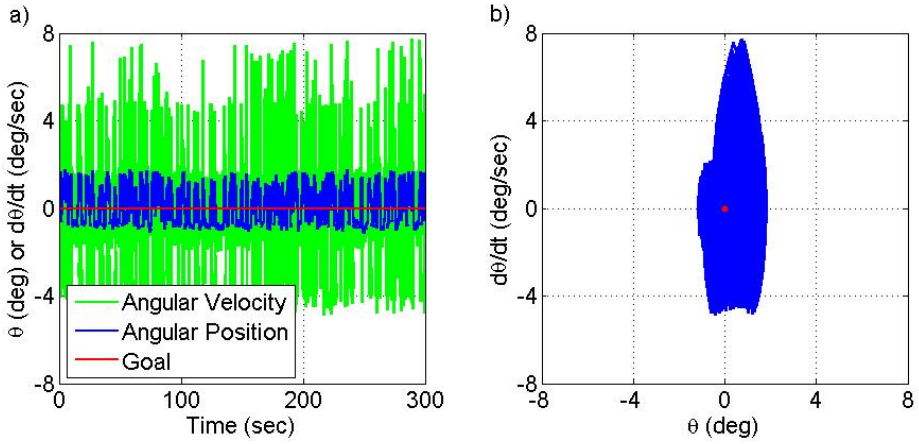

Fig. 19. Basic Inverted Pendulum Controller, 300 Seconds: Time History (a) and Phase Diagram (b)

Initial attempts at integrating AAG into this problem had the goal regions dependent solely on θ as is used in Case 1. Simulations using the learned controller were not satisfactory, so a more complex scheme was implemented. This scheme is shown in Figure 20. For this case the goal regions are dependent on both θ and $\dot{\theta}$. The equations that describe the bounds shown in Figure 20 are listed in Table XII. Figure 20 also shows all of the quantized states where information was stored by the agent. When learning, these bounds also act as the outer bounds at finer levels of discretization. For example, the first level of quantization uses the outer bounds listed in Table VII and the goal bounds listed in Table XII. For the second level of quantization, the linear equations describing the goal region for the first level of quantization now act as the outer bounds for the second level, and the goal region for the second level of quantization is in effect.

The value function and policy for this case are shown in Figure 21. The actions for the policy are the same as the previous case and are listed in Table X. The value function for this case shows a more extreme peak at $\left(\theta, \dot{\theta}\right) = (0, 0)$ and a similar crest

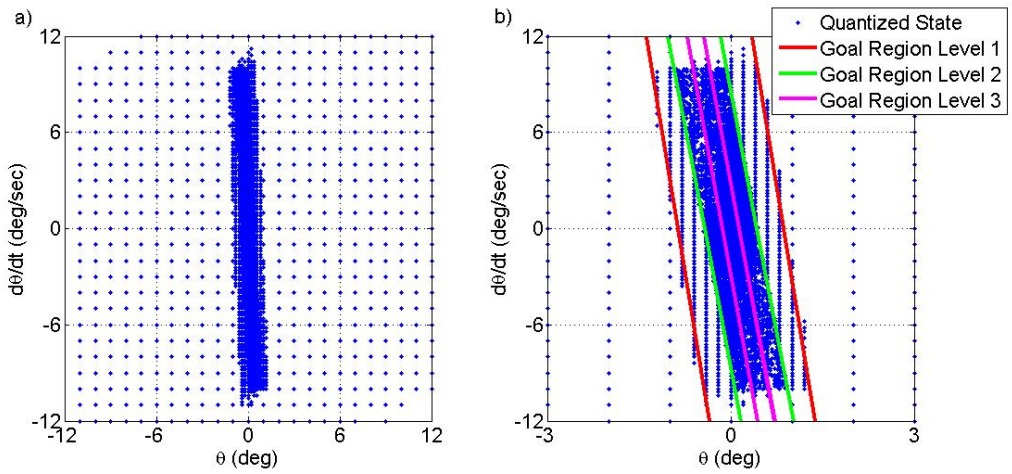

Fig. 20. AAG Inverted Pendulum Controller: Quantized States (a) and Goal Regions (b)

Table XII. Inverted Pendulum AAG Goal Region Equations

Discretization Level	Lower Bound Equation	Upper Bound Equation
1	$\dot{\theta} = -23.3\theta - 20$	$\dot{\theta} = -23.3\theta + 20$
2	$\dot{\theta} = -20\theta - 8.6$	$\dot{\theta} = -20\theta + 8.6$
3	$\dot{\theta} = -20.7\theta - 2.9$	$\dot{\theta} = -20.7\theta + 2.9$

in the goal region. The levels of quantization are also evident in the value function in the blocky regions on the outer edges of the crest slope and beyond and the fine details near the center of the crest and the peak. The visual representation of the policy again shows that the agent probably needs more learning time. However, the regions of a given action are more contiguous than the previous case. The patterning of the policy suggests that good control of the pendulum when exploiting learned knowledge is sensitive to initial conditions.

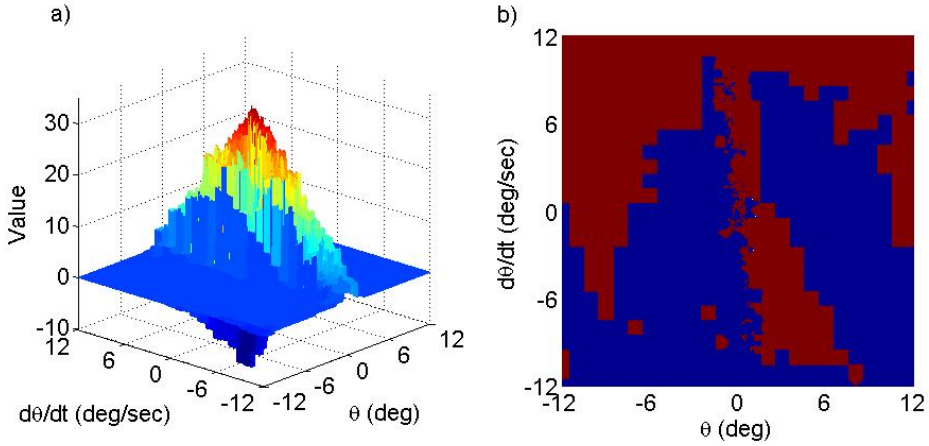

Fig. 21. AAG Inverted Pendulum Controller: Value Function (a) and Policy Representation (b)

This controller is also tested for a short time period and an extended time period as shown in Figures 22 and 23, respectively. The controller is implemented such that the actions available to the controller is dependent on the state, as it was during learning. This means that if the state of the pendulum is outside the level 1 goal region bounds, then the agent may only use the level 1 actions. When the state of the pendulum is within the level 1 or level 2 goal region bounds, the controller may use the level 2 or level 3 actions, respectively. If the state is within the level 3 goal region bounds, then the controller may use the level 3 actions. The time history in

Figure 22 shows that this controller is able to balance the pendulum much closer to the equilibrium point than the previous controller. The agent balances to within an angle of ± 0.3 deg within 5 seconds of the simulation. The angular velocity shows clearly the effects of the controller. In the previous case the angular velocity time history was very jagged as the controller sought to balance the pendulum by alternating between two large opposing forces. The angular velocity shown in Figure 22 shows evidence of the controller using the smaller forces, which results in a smoother time history and better control of the pendulum. The angular velocity does not get as large as in the previous case. The largest spike in angular velocity is 3.3 deg / sec and quickly reduces to within ± 0.5 deg / sec. All of these details are also shown in the phase diagram. The small red dot indicates the ideal goal of a perfectly balanced pendulum. The phase diagram shows that the pendulum oscillates briefly before good balance is achieved near this point.

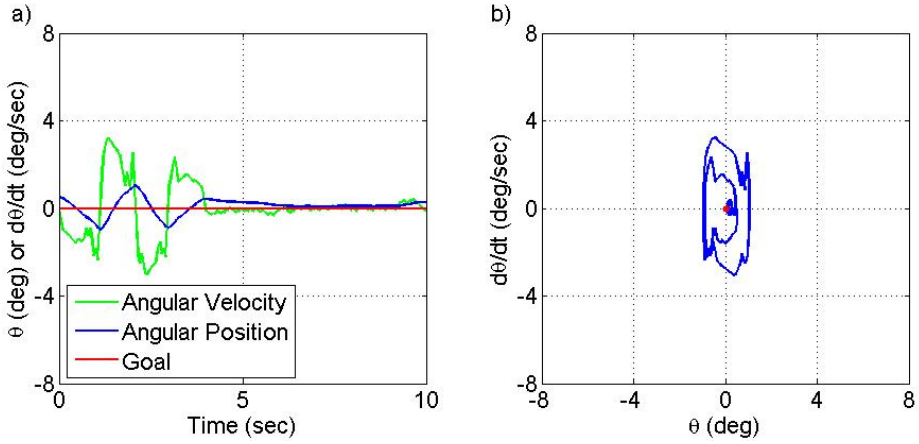

Fig. 22. AAG Inverted Pendulum Controller, 10 Seconds: Time History (a) and Phase Diagram (b)

The time history in Figure 23 shows that the same trend continues as time elapses. The controller maintains balance of the pendulum to within an angle of

± 0.3 deg for the duration of the 300 sec. The angular velocity also remains low, within ± 0.5 deg / sec, after the initial balancing for the duration of the simulation. On closer inspection of the time history and phase diagram, it is evident that the pendulum is near the equilibrium, but tends unbalance to a positive angle, though it quickly recovers. This trend is due to the velocity of the cart, which is not considered in this formulation. The cart velocity stabilizes around $x = 5m/\,\text{sec}$, which in turn causes the pendulum to drift to a positive angle. The current controller is able to balance the pendulum despite this fact, but it could be avoided by putting constraints on x and $\dot{x}$ and including them in the learning problem, thereby creating a more complex problem.

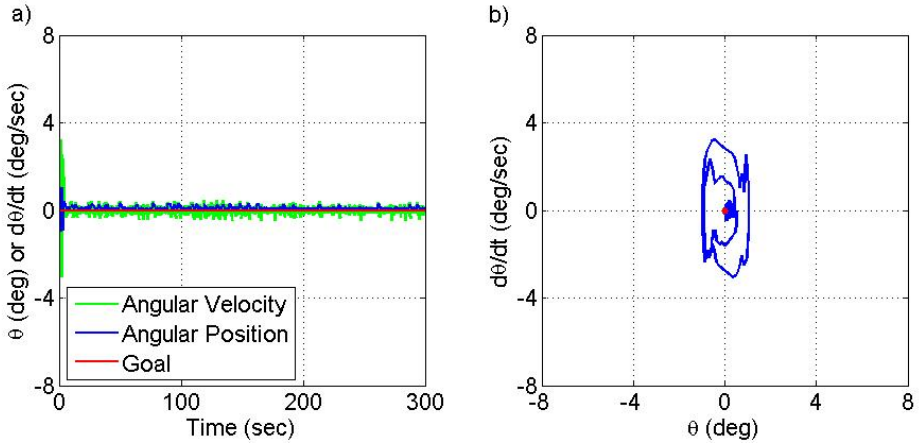

Fig. 23. AAG Inverted Pendulum Controller, 300 Seconds: Time History (a) and Phase Diagram (b)

D. Summary

In this chapter it was cast a reinforcement learning problem so that both Q-learning and the multi-resolution state-space discretization method could be tested. The value

function and greedy policy were visually analyzed and then tested in simulation. Results show that controller learned just using Q-learning was able to balance the pendulum within the specified range of $|\theta| \leq 2\,\text{deg}$, but could not damp out the oscillations. The angular position never got closer than $\theta = -1\,\text{deg}$ and $\theta = 2\,\text{deg}$ with the angular velocity approaching $8\,\text{deg}\,/\,\text{sec}$ many times. The controller learned using Q-learning and multi-resolution state-space discretization was able to balance the pendulum much nearer to the equilibrium point without the extremes in angular velocity. The agent balanced the pendulum to within an angle of $\pm 0.3\,\text{deg}$ within 5 seconds, and the angular velocity remained $< 0.5\,\text{deg}\,/\,\text{sec}$ for most of the simulation. This balance was maintained for the 6000 time steps of the 300 sec simulation.

CHAPTER VIII

SIMPLE RECONFIGURABLE SYSTEM EXAMPLE - MORPHING AIRFOIL

When considering a reconfigurable air vehicle, the complexity of the reconfiguration can range from changing a few parameters, such as wing dihedral, wing sweep, wing span, the airfoil itself, *etc.*, to a fully articulated wing, body, and tail much like a bird's. Aircraft are usually designed for very specific purposes. As my advisor likes to say, "Form follows function." A typical fighter aircraft (Figure 24) has a highly swept, low aspect ratio wing and is designed for speed and maneuverability in flight. A bomber aircraft is designed for efficiency in cruise and to carry a large ordinance

Fig. 24. Representative Fighter Aircraft - F-16 Fighting Falcon

payload. Thus they tend to be large with a slightly swept, high aspect ratio wing (Figure 25). A general aviation (GA) aircraft is designed for stability and to fly at slower speed, $< 150kts$ (Figure 26).

The benefit of a reconfigurable aircraft lies in the ability to harness the capabilities of many types of aircraft. Conceptually, such a vehicle would be able to perform the duties of a fighter, bomber, GA, *etc.* as well as or better than those aircraft

Fig. 25. Representative Bomber Aircraft - B-1B Lancer

Fig. 26. Representative General Aviation Aircraft - Cessna 172

designed specifically for those purposes. Thus, as described in the Chapter I, this is an active research area.

Many current approaches involve optimizing the vehicle shape for several flight phases, applying an optimal control technique of one form or another, and determining an actuation scheme to enable the shape change. Some of the problems that arise with this approach are that there are only a handful optimal shapes and the optimized controller is specific to the initial and final shapes. Should the optimal configuration be redesigned, that would require that the shape change controller be redesigned as well. Machine learning and in particular the algorithm developed in Chapters III-VI of this dissertation is a candidate approach to avoid these problems.

The first application of this algorithm to a reconfigurable system is a simple airfoil as seen in Figure 27. Rather than finding a couple optimal shapes that meet some criteria and a separate optimal controller to maneuver from one shape to the other, the airfoil is cast as a reinforcement learning problem in which the full spectrum of shapes that meet the flight phase criteria are learned as well as the local transitions that guide the shape from *any* initial shape to a shape that meets the requirements.

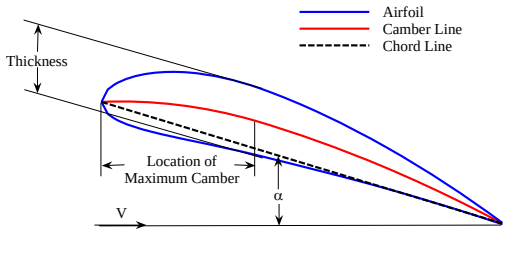

Fig. 27. Representative Airfoil

Section A discusses the airfoil model to be cast as the environment of the reinforcement learning problem. Then the full reconfigurable airfoil reinforcement learn-

ing problem is described followed by an extensive series of examples that exercise the many aspects of the problem. Several developmental stages are shown in Section C. The numerical results for this application are much more extensive than the other applications because the various components of MGAP were developed with the reconfigurable airfoil as the primary testbed.

A. Airfoil Model

The airfoil is modeled by a CFD code using a constant strength doublet panel method. This model calculates the aerodynamic properties of an airfoil given a set of four inputs:

- Airfoil thickness
- Airfoil camber
- Location of maximum camber
- Airfoil angle-of-attack

Setting these four parameters as inputs to the model allow for quick calculation of the aerodynamic properties of many different airfoils or as a single airfoil changes shape.

A number of assumptions are made during the development of the model. These assumptions are

1. Flow is incompressible.
2. Flow is inviscid.
3. Upper and lower surfaces of the airfoil are pinned at the leading and trailing edge.

These assumptions allow for a simple model that is valid for the linear range of angle-of-attack and sufficient for the purposes of this research.

Calculating the aerodynamic properties of the airfoil in question begins with considering the horizontal, u_p, and vertical, w_p, velocities of each panel in the local panel coordinate system.

$$u_p = \frac{\mu}{2\pi}\left[\frac{z}{(x-x_1)^2+z^2} - \frac{z}{(x-x_2)^2+z^2}\right] \tag{8.1}$$

$$w_p = \frac{\mu}{2\pi}\left[\frac{x-x_1}{(x-x_1)^2+z^2} - \frac{x-x_2}{(x-x_2)^2+z^2}\right] \tag{8.2}$$

which requires a transformation of the global coordinate system to the local panel coordinate system using the following relationship:

$$\begin{pmatrix} x \\ z \end{pmatrix}_p = \begin{pmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{pmatrix} \begin{pmatrix} x - x_0 \\ z - z_0 \end{pmatrix} \tag{8.3}$$

Assuming that there is no penetration of the boundary, which means that the flow cannot cross the boundary of the airfoil, the velocity of the flow normal to the surface is 0 in the global coordinate system. Transforming the velocities into the global coordinate system is thus achieved with Eq. 8.4.

$$\begin{pmatrix} u \\ w \end{pmatrix} = \begin{pmatrix} \cos(\theta_i) & \sin(\theta_i) \\ -\sin(\theta_i) & \cos(\theta_i) \end{pmatrix} \begin{pmatrix} u_p \\ w_p \end{pmatrix} \tag{8.4}$$

The doublet strengths, μ, and resulting tangential velocities can then be solved for from a system of equations for every panel constructed from Eqs. 8.1-8.4. These tangential velocities are then used to calculate the pressure coefficient using a modified form of the Bernoulli equation:

$$C_p = 1 - \frac{u^2 + w^2}{V_\infty^2} \tag{8.5}$$

The pressure coefficient can be broken up into normal and axial forces using simple integration. These forces can also be further broken up into lift and drag using simple trigonometry.

$$C_n = \frac{1}{c}\int_0^c \left(C_{p_{lower}} - C_{p_{upper}}\right) dx \tag{8.6}$$

$$C_a = \frac{1}{c}\int_0^c \left(C_{p_{upper}}\frac{dy_{upper}}{dx} - C_{p_{lower}}\frac{dy_{lower}}{dx}\right) dx \tag{8.7}$$

$$C_l = C_n \cos(\alpha) - C_a \sin(\alpha) \tag{8.8}$$

$$C_d = C_n \sin(\alpha) + C_a \cos(\alpha) \tag{8.9}$$

Validation and verification of this model can be found in Ref. [183].

B. Airfoil Cast as a Reinforcement Learning Problem

Casting the morphing airfoil as a reinforcement learning problem is a slightly different challenge than the inverted pendulum in Chapter VII. In the case of the airfoil, there are no dynamics involved in the shape change. For the purposes of this research, each commanded change in airfoil shape yields an immediate response. This formulation effectively removes a potentially complicated aspect of the morphing airfoil problem from this reinforcement problem and allows for better focus on the actual choice in shape.

The state variables must be chosen just as carefully, though, so that the aerodynamic properties of the airfoil are adequately exploited. The CFD model itself constitutes the *environment* with which the agent interacts. Thus there are four possible interdependent parameters, the four inputs to the CFD model, that can constitute the state of the agent within the environment. The reinforcement learning

problem can be set up such that any combination of the four parameters form the state while the others are just held constant in the background.

The agent interacts with its environment by choosing actions from a set of admissible actions. The state-space is discretized in the manner described in Chapter III, so these actions include incremental changes in the shape parameters of the airfoil. Thus the agent is effectively restricted to movement between adjacent vertices. An example of admissible action in this context is the following. The agent chooses to move in the x_1-direction from vertex ${}^{IJ}X$ in the 2-dimensional problem. For the initial discretization, the two possible actions in the x_1-direction are defined as follows

$$\begin{aligned} A^1_{11} &\equiv {}^{(I+1)J}X - {}^{IJ}X = h^1_{x_1} \\ A^1_{12} &\equiv {}^{(I-1)J}X - {}^{IJ}X = -h^1_{x_1} \end{aligned} \tag{8.10}$$

Eq. 8.10 can be summarized by saying the initial admissible actions in the x_1-direction are $A_1 = \pm h^1_{x_1}$. Similar relationships can be found for the x_2-direction. Admissible actions in the other direction is $A_2 = \pm h^1_{x_2}$. The definitions of the x_i axes for all of the examples are defined in Table XIII. To read these tables consider the x_1-direction, for example. The agent changes $\pm 0.50\%$ of the chord in thickness in this direction when $h_{x_1} = 0.50\%$.

The *goal*, g, of the agent for this problem is defined by the aerodynamics of the airfoil. Every goal has a range, g_r, associated with it. The majority of the numerical examples have goals defined by the airfoil lift coefficient, c_l. A couple examples have goals based on airfoil drag, c_d, and moment, c_m, coefficients. Those vertices whose state yield aerodynamic coefficients, c, from the CFD model that lie in the goal range defined by $g - g_r \leq c \leq g + g_r$ form the pseudogoals of the problem.

Table XIII. Morphing Airfoil Axis Definitions

x_i	Definition
x_1	Thickness (%)
x_2	Camber (%)
x_3	Location of Maximum Camber
x_4	Angle-of-Attack (deg)

Table XIV. Morphing Airfoil Q-Learning Example Reward Structure

Bounds	Reward
$g - g_r \leq c \leq g + g_r$	20
$s < limits_{min}$ or $s > limits_{max}$	-20
$Otherwise$	0

There are two reward structures used by the various examples. The first is a traditional negative, neutral, and positive reinforcement scheme and was used in the early stages of this research. This scheme is summarized in Table XIV. The $s < limits_{min}$ or $s > limits_{max}$ refers to when the state, s is beyond the bounds or limits of the of the state-space. These limits are listed in Table XV. The second is a form of reward shaping that is essentially a gradient based reward function. The function is defined by Eq. 8.11.

$$r = |g - c_{n-1}| - |g - c_n| \tag{8.11}$$

where r is the reward, g is the goal, and c is the metric or aerodynamic coefficient.

Table XV. Morphing Airfoil Parameter Limits

Limit	Lower	Upper
Thickness (% chord)	10	18
Camber (% chord)	0	5
Location of Max Camber	0.2	0.8
Airfoil Angle-of-Attack (deg)	-5	5

C. Numerical Results

In each numerical example the agent is given the task of learning to maneuver through the state-space to a goal state that satisfies some aerodynamic requirement. The agent is usually allowed 5000 episodes over which to learn unless otherwise specified. The initial state for each episode is random.

For this example, the agent learns using various combinations of the components described in previous chapters. Section 1 describes the results of learning using just Q-learning. Shape changing of both two and four shape parameters is explored. Section 2 is the first instance in which AAG was originally tested. Several developmental stages were involved before the final form of that part of the algorithm was realized. Section 3 has the agent learn using both AAG and the policy comparison stopping criteria based on the policy as developed in Chapter V. Finally, Section 4 is the full MGAP algorithm as applied to the morphing airfoil problem. All of these examples show the difference the various components make when they are incorporated into the basic Q-learning algorithm.

1. Case 1: Q-Learning

There were several developmental stages using just Q-learning during the early stages of casting the morphing airfoil problem as a reinforcement learning problem. The first stage has the agent learn several goals based on combinations of the aerodynamic coefficients with two state variables. The second stage tests several of the parameters in the Q-learning algorithm itself, such as learning rate, discount factor, and policy followed, also with two state variables. The final stage gives the agent full rein of the four shape parameters as the state variables. These three stages serve to show the versatility of basic Q-learning as it is applied to the morphing airfoil reinforcement learning problem.

a. Developmental Stage 1: Initial Testing

The morphing airfoil reinforcement learning problem is first tested by evaluating several reward schemes. The purpose of these examples is to show that the Q-learning agent can explore, learn, and use its knowledge of the state-space defined by the aerodynamic model. The agent explores over the state-space defined by the state variables of airfoil thickness and airfoil camber. Three cases with different aerodynamic requirements are learned. The first case has the goal defined by only the lift coefficient. The agent is looking for a configuration that meets a minimum lift coefficient requirement. The second case adds complexity by having the agent also try to meet a maximum drag requirement. This additional requirement effectively narrows the goal region previously defined by the minimum lift coefficient requirement. The third case adds one more requirement in the form of a maximum moment coefficient about the leading edge of the airfoil, further reducing the goal region. The requirements for each case are enumerated in Table XVI.

Table XVI. Reward Region for Morphing Airfoil Cases 1-3

	Case #1	Case #2	Case #3
c_l	≥ 0.4	≥ 0.4	≥ 0.4
c_d	N/A	≤ 0.003	≤ 0.003
$c_{m_{le}}$	N/A	N/A	≤ 0.87

Table XVII. Learning Parameter Constants for Morphing Airfoil Developmental Stage 1

Parameter	Value
Chord	$1m$
Angle-of-attack	$2.0°$
Episodes	5000
α	0.01
γ	0.7

For the purpose of direct comparison, the chord, angle-of-attack, and number of episodes are kept constant and are listed in Table XVII. The reward scheme is that defined above by Table XIV. For each of the 5000 episodes, the agent begins in a random initial state that is not classified as a goal state. It explores the state-space of thickness-camber combinations until it hits the predefined limit of total number of actions or finds a goal state. Should the agent run into a boundary, that boundary location is noted, and the agent chooses another action. The actions for all three cases are defined by the h_{xi} listed in Table XVIII.

Table XVIII. Distance Between Adjacent Vertices for Morphing Airfoil Developmental Stage 1

h_{x_i}	Value
h_{x_1}	0.10
h_{x_2}	0.10

Case #1: Lift Coefficient Goal: This case tests the learning algorithm in which the goal is defined by a minimum lift coefficient the agent must find based on the aerodynamic calculations given the current state's thickness-camber pair. The goal region is therefore all of the pairs that meet the minimum lift coefficient requirement of 0.4.

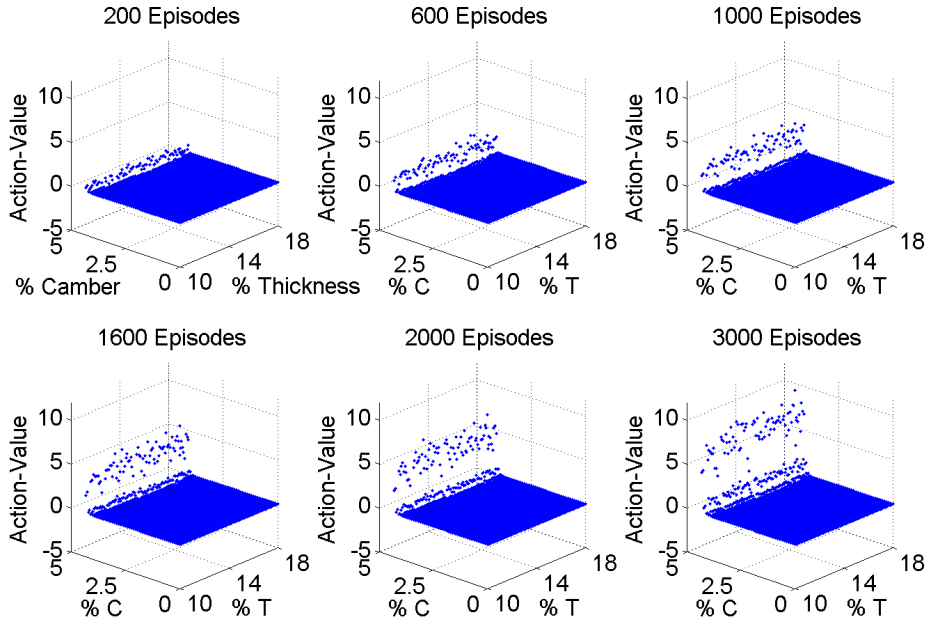

Fig. 28. $Q(s, a)$ Evolution for an Increase in Camber for Case #1

Figure 28 shows the evolution of $Q(s, a)$. Given the 4-dimensional nature of $Q(s, a)$ (thickness, camber, action, action-value) only one action can be displayed at a time. Figure 28 illustrates the action-value for each thickness-camber pair for the action of an $+0.10\%$ increase in camber as the number of episodes increases. Very early on the agent learns that the goal region encompasses higher values of camber as indicated by the positive values of the action-value function. These positive values indicate a positive preference for this action illustrated. If this preference is greater than the preferences for the other three actions, then the agent will choose this action given that it is acting in a greedy manner. This preference becomes more pronounced as the number of episodes increases. It is also apparent that as the number of episodes increases the surface suggested by the values becomes more fleshed out. The action-value for pairs of somewhat lower thickness becomes more positive. This trend indicates that agent is learning that at these lower camber values if it chooses to increases camber it will get closer to the goal region. Finally, Figure 28 shows that at any given thickness, there is at least one value of camber that is in the goal region.

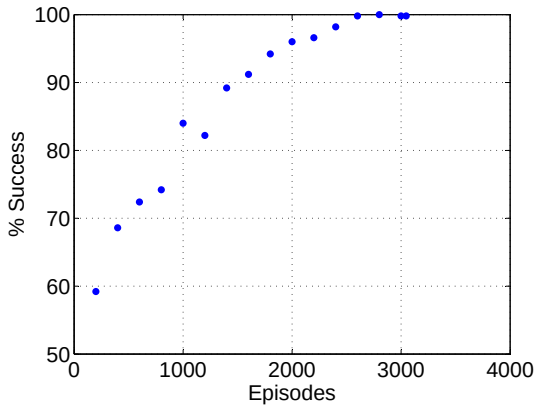

Fig. 29. Monte Carlo Simulation for Q-learning of Case #1

The Monte Carlo simulation for performance is conducted on this learning case, though it was not used as a stopping criterion. The learning is paused every 200 episodes and the performance of the policy measured as described in Chapter V. Figure 29 shows the trend of success as the number of episodes increases. The percent success increases rapidly and approaches 100% success, meaning that the agent navigated to a goal state in the goal region every episode. Figure 29 indicates that by using this learning algorithm, the agent is successful 90% of the time by 1600 episodes and approximately 100% of the time by 2600 episodes given the definition of the goal for this example. There is a slight decrease in success from 1000 to 1200 episodes. This occurrence is most likely due to the occasionaly random actions taken by agent resulting in the agent encountering boundaries more often than in the preceding and succeeding cases.

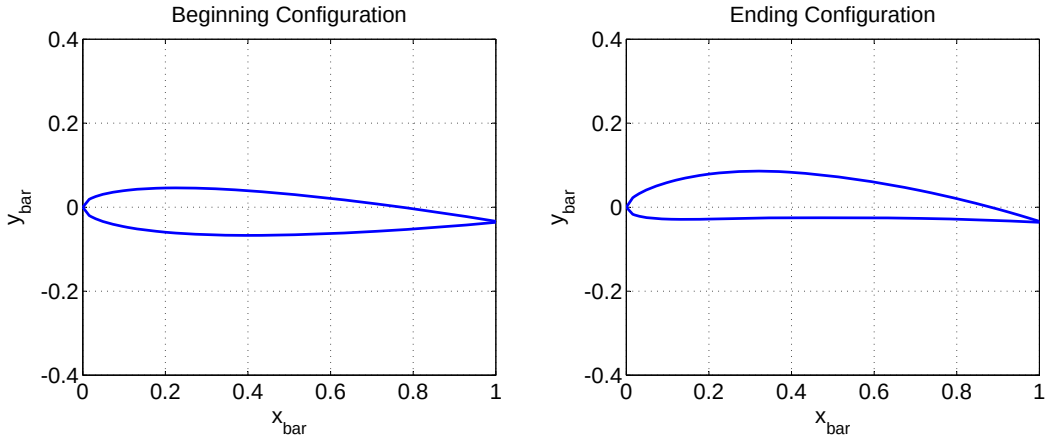

Fig. 30. Initial and Final Airfoil Configuration for Case #1

Figures 30 and 31 illustrate one successful episode using the final learned action-value function. The agent's initial state is 11% thickness and 0% camber. It uses the learned function to navigate from this initial state to a state in the goal region.

The agent chooses an action every 0.5 sec based on its current state defined by its thickness and camber using the action-value function it learned offline; either the thickness or camber changes according to arbitrarily chosen simple nonlinear shape changing dynamics. Thus the airfoil changes from one static shape to another by a series of small steps in an effort to reach the defined goal. Figure 30 shows the airfoil configuration for the initial and final state. Figure 31 shows how the thickness and camber change with time. The resulting lift coefficient in Figure 31 shows that the agent chooses actions that takes it steadily toward the lift coefficient goal of 0.4 until it reaches a thickness and camber combination that corresponds to a calculated lift coefficient greater than 0.4.

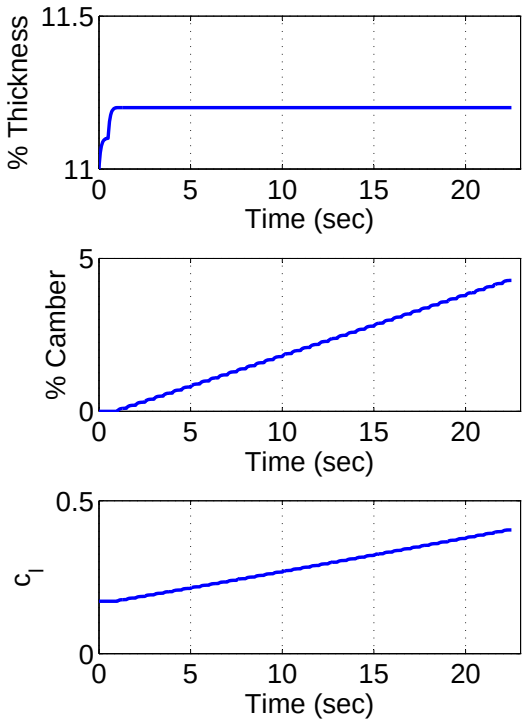

Fig. 31. State Progression Using Greedy Policy for Case #1

Case #2: Lift and Drag Coefficient Goal: This case tests the learning algorithm in which the goal is defined by a minimum lift coefficient and a maximum drag coefficient the agent must find based on the aerodynamic calculations. The goal region is all of the pairs that meet the minimum lift coefficient requirement 0.4 and the maximum drag coefficient requirement of 0.003 as defined in Table XVI.

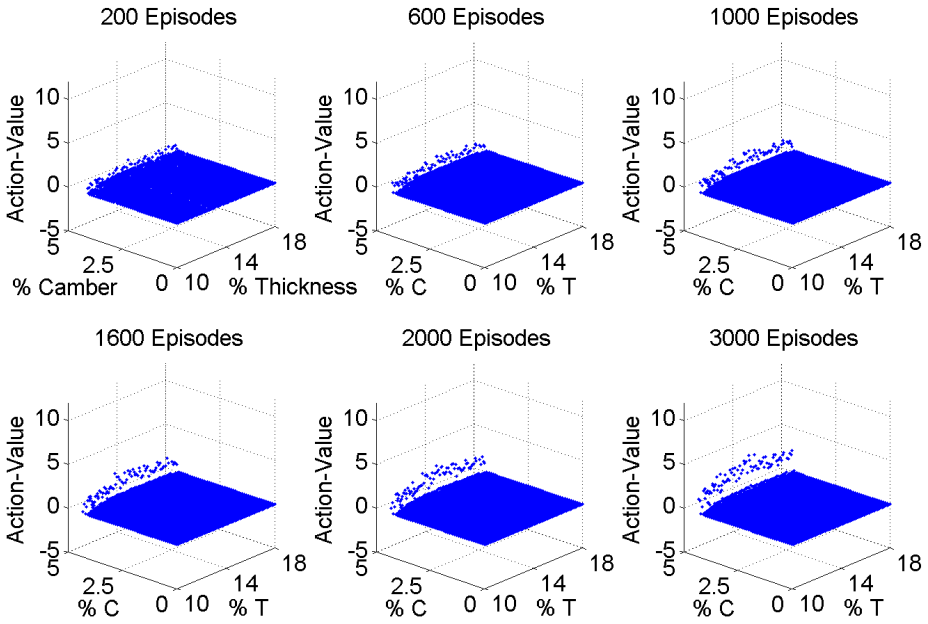

Fig. 32. $Q(s, a)$ Evolution for an Increase in Camber for Case #2

Figure 32 shows the evolution of $Q(s, a)$ for the action of a +0.1% increase in camber for this goal definition. Again the agent learns that the goal region encompasses higher values of camber. When compared to Figure 28, it can be seen that for this goal definition the goal region is much narrower than previously. The addition of the drag coefficient requirement restricts even further the possible thickness-camber pairs that satisfy the goal requirements. Also, as the number of episodes increases,

the surface for this action becomes more pronounced, though not as rapidly as the previous example. Like the previous example, however, Figure 32 shows that at any value of thickness, there is at least one value of camber such that the pair meets the goal requirement.

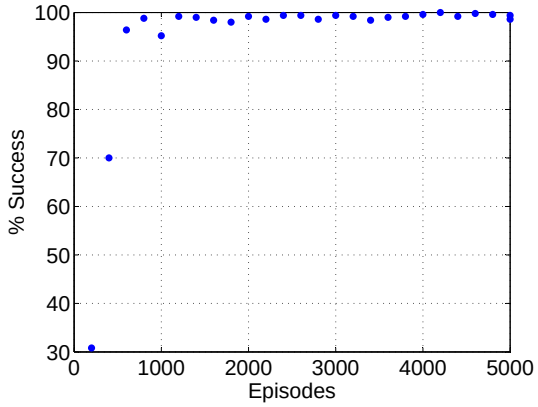

Fig. 33. Monte Carlo Simulation Q-Learning of Case #2

Figure 33 displays the results of the Monte Carlo simulation for this example. The success rate approaches 100% more rapidly than in the previous example. Figure 33 indicates that using this learning algorithm with the more restricting goal requirement, the agent is successful 90% of the time by 600 episodes approximately 100% successful by 2000 episodes. The small oscillations in success rate as the number episodes increases and the marked decrease in success from 800 to 1000 episodes is most likely due to the occasional random actions taken by the agent that resulted in a cessation of the current episode.

Figures 34 and 35 illustrate one successful episode using the final learned action-value function given the goal requirements for this example. The initial state is 11% thickness and 0% camber. It uses the learned function for this example to navigate

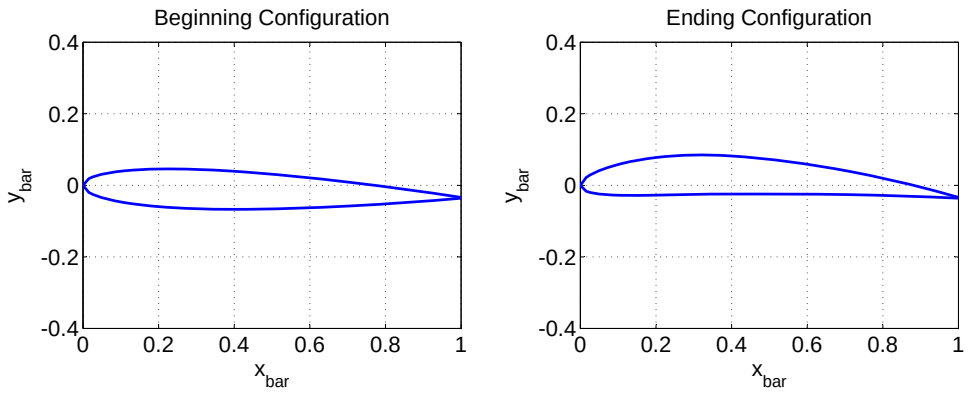

Fig. 34. Initial and Final Airfoil Configuration for Case #2

to the goal region without encountering a boundary. Figure 34 shows the initial and final configuration of the airfoil, and Figure 35 shows how the agent navigates from one to the other. The agent chooses actions that takes it directly toward a state that has a lift coefficient greater than 0.4 and a drag coefficient less than 0.003. In this case the agent does not choose to change thickness at all. It instead prefers to increase camber steadily until the goal region is reached. The lift coefficient and drag coefficient calculated by the aerodynamic module reflect this as they approach their respective goals as a result of the actions the agent chooses.

Case #3: Lift, Drag, and Moment Coefficient Goal: This case further tests the learning algorithm by restricting the goal region even more by adding a maximum moment coefficient about the leading edge the agent may not exceed in addition to the minimum lift coefficient and a maximum drag coefficient the agent must adhere to. The goal region is thus all the pairs that meet the minimum lift coefficient requirement 0.4, the maximum drag coefficient requirement of 0.003, and the maximum moment coefficient about the leading edge as defined in Table XVI.

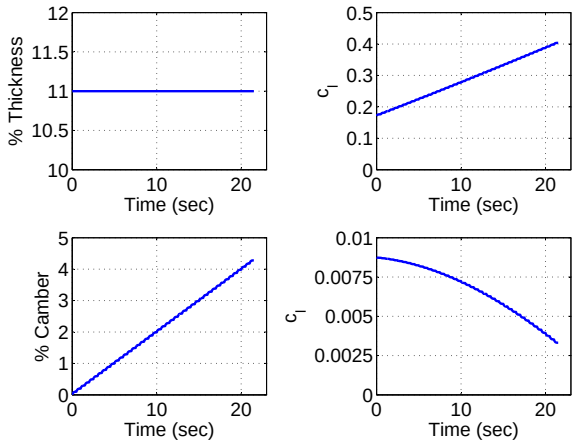

Fig. 35. State Progression Using Greedy Policy for Case #2

Figure 36 depicts the evolution of $Q(s, a)$ for the action of an +0.1% increase in camber for this goal definition. The contour of the surface for this goal definition is much different than in the previous two examples. There are two small regions that meet all the requirements defined in Table XVI. The locations of the two become more apparent as the positive preferences become more pronounced as the number of episodes increases. The smaller region is centered near 12% thickness and 4.7% camber. The larger region is in the corner of large percent thickness and large percent camber. This figure also shows the result of the agent encountering a boundary - negative action-values. These negative preferences tell the agent that a boundary is near and it should choose some other action.

Despite this smaller goal region, the success rate shown in the Monte Carlo simulation results in Figure 37 approaches 100% slowly as the number of episodes increases, similar to the previous examples. The learning algorithm is able to cope with the more restricting goal requirement reaching 90% success rate by 1400 episodes. The agent is almost 100% successful by 1600 episodes. Similar to the previous examples, there are

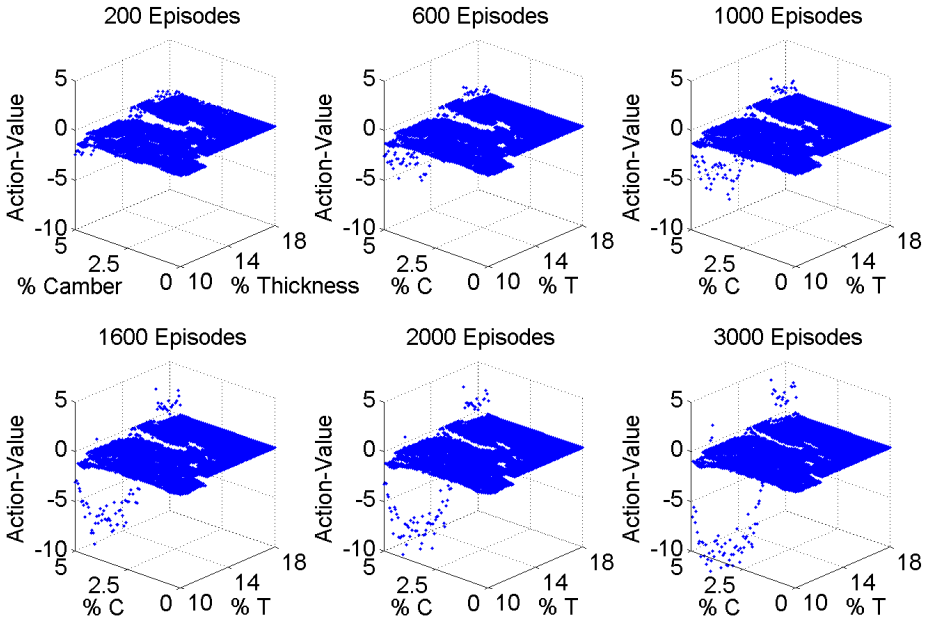

Fig. 36. $Q(s, a)$ Evolution for an Increase in Camber for Case #3

small oscillations just below 100% success as the number of episodes increases. These oscillations are possibly due to the occasional random action the agent takes. Given the smaller goal region there is more exposed boundary, and therefore the agent is more likely to take a random action and encounter that boundary. Figure 37, however, shows that this adverse effect on performance is within acceptable bounds.

Figures 38 and 39 illustrate one successful episode using the final learned action-value function given the goal requirements for this example. The initial state is 11% thickness and 0% camber as in the previous two exmples. It uses the learned function for this example to navigate to the much smaller goal region without encountering a boundary. Figure 38 shows the initial and final configuration of the airfoil, and Figure 39 shows how the agent navigates from one to the other. It takes a longer period of

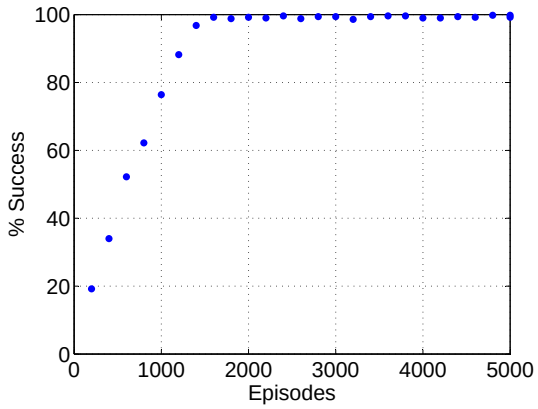

Fig. 37. Monte Carlo Simulation for Q-Learning of Case #3

time for the agent to reach the larger of the two goal regions, almost twice that of the previous two examples. The reason is that it must choose to change both thickness and camber many times to traverse the distance from the initial state to a goal state. These choices are shown in Figure 39 as the agent alternates between choosing to change thickness and choosing to change camber. This figure also shows the changes in lift, drag, and moment coefficient as the agent navigates to a configuration that meets all of the defined requirements.

b. Developmental Stage 2: Q-Learning Parameter Testing

The morphing airfoil reinforcement learning problem using just Q-learning is now tested by investigating the effects of the various learning parameters, namely the discretization of the state-space that define the action sets, the discount factor, and policy followed. The learning performance is analyzed in several ways. First the dimensionality of the action-value functions are compared between the action sets. Next the Monte Carlo simulation results for different policies are compared for each set

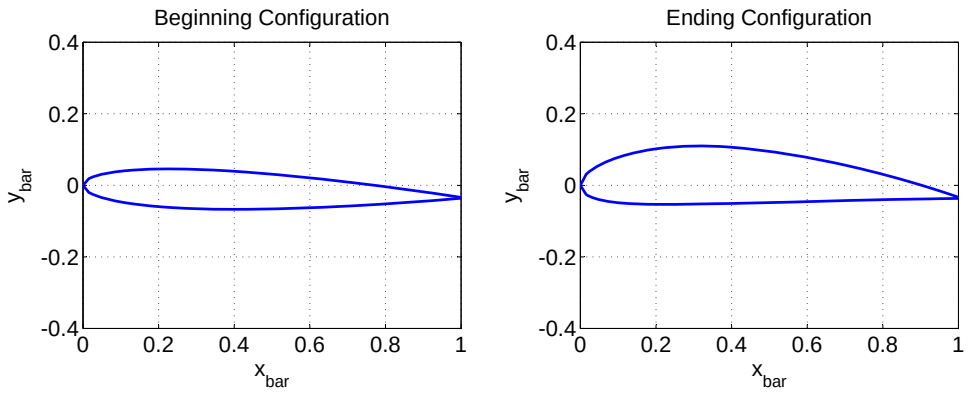

Fig. 38. Initial and Final Airfoil Configuration for Case #3

Table XIX. Distance Between Adjacent Vertices for Morphing Airfoil Developmental Stage 2

Cases	h_{x_1}	h_{x_2}
Case 1	0.10	0.10
Case 2	0.25	0.25
Case 3	0.50	0.50
Case 4	1.00	1.00

of h_{x_i} to examine both the learning performance for a particular set of h_{x_i}. The Monte Carlo simulation results are then recast such that comparisons between sets of h_{x_i} for each policy can be more easily made to again examine the learning performance. The final value functions are then compared and analyzed.

Dimensionality: The dimensionality of the problem to be learned is an ever present concern for learning algorithms. A problem with a high number of states, or in the case of Q-learning, a high number of state-action pairs, is that it necessitates

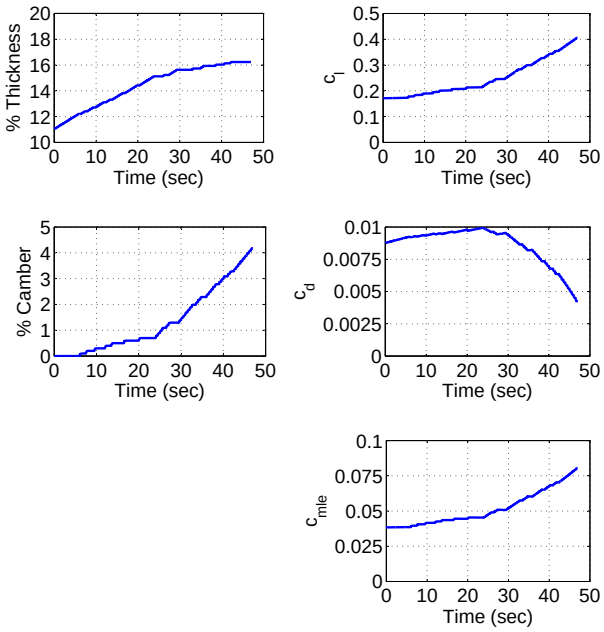

Fig. 39. State Progression Using Greedy Policy for Case #3

a greater number of learning episodes, and thus more computational time, to learn the required action-value function. Conversely, fewer states means faster learning, especially given that computational aerodynamic models are usually computationally intensive. However, there must be enough states to fully capture the details of the action-value function for the problem at hand. The h_{x_i} for each case are defined in Table XIX. The associated number of states and state-action pairs for each value of h_{x_i} are listed in Table XX.

Policy Comparison and Analysis: This section considers a direct comparison for each h_{x_i} between action-value functions when the agent follows one of the following:

- 100% exploration policy

Table XX. Non-Goal States and State-Action Pairs for Morphing Airfoil Developmental Stage 2

h_{x_i}	States	State-Action Pairs
0.10	3379	13516
0.25	530	2120
0.50	153	612
1.00	45	180

- ε-greedy policy with annealing
- ε-greedy annealing policy and annealing of the discount factor, γ

The results of the simulations described above are shown in Figure 40. All four subfigures show that the action-value function in which the agent followed an annealing policy converges to a good solution the fastest. A $96\% - 100\%$ success rate is achieved in as little as 200 episodes for the coarsest discretization and 800 episodes for the finest discretization. The results of the learning in which the discount factor also anneals shows a much lower success rate early in the learning then rapid convergence after a couple hundred episodes. In this case, convergence to a $95\% - 100\%$ success rate is achieved in $200 - 3000$ episodes. The most distinct differences between policies are seen in Figure 40.a. Rate of convergence is expected to be lower given the larger number of states the agent must visit, but Figure 40.d suggests that when the agent must only visit a small number of states, annealing the discount factor and/or the policy makes little difference in the rate of convergence. The "kink" evident in Figure 40.b is most likely a result of the small probability that the agent chooses a random action and encounters a boundary.

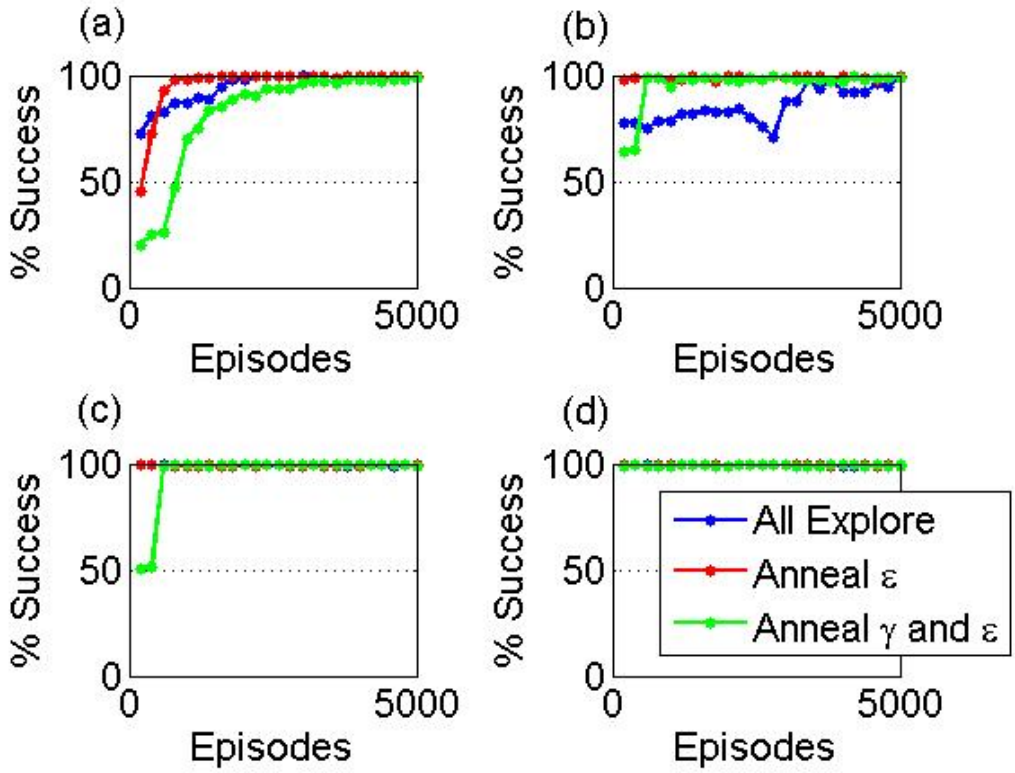

Fig. 40. Policy Comparison for Each h_{x_i}: a) $h_{x_i} = 0.10\%$, b) $h_{x_i} = 0.25\%$, c) $h_{x_i} = 0.50\%$, and d) $h_{x_i} = 1.00\%$

h_{x_i} Comparison and Analysis: This section takes a different approach than the previous section and considers a direct comparison between the action-value functions for each h_{x_i}. The results presented in this manner are shown in Figure 41. This figure shows that there is the most difference in convergence between h_{x_i} values when the policy is fully explorative and when both the policy and discount factor changes as learning progresses. 5000 and 3000 episodes, respectively, are required for all four action-value functions to converge to at least a 95% success rate. Generally, convergence for these cases increases as h_{x_i} increases. This result is again due to the decreasing number of states that the agent must visit and update. Figure 41.b

shows the highest rate of convergence for all values of h_{x_i}. All four action-value functions converge to nearly a 100% success rate within 800 episodes. This result suggests that for this problem annealing the policy from fully explorative to almost fully exploitative yields the best results.

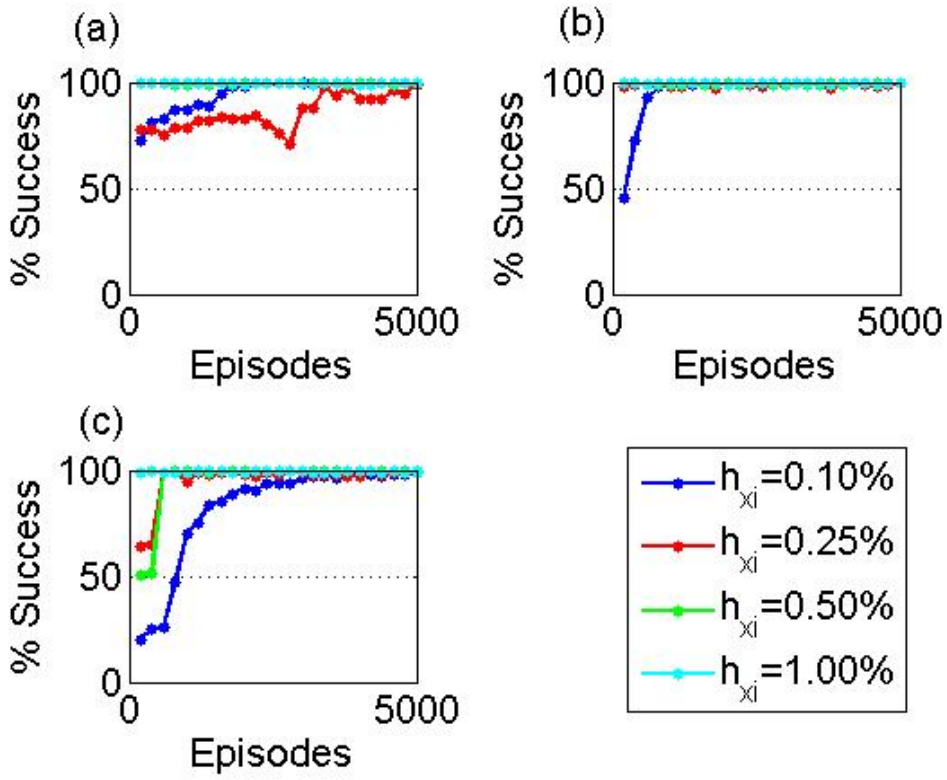

Fig. 41. h_{x_i} Comparison: a) 100% Exploration, b) Anneal ε, c) Anneal ε and γ

Value Function Analysis: It is helpful to consider a value function approximated from the learned action-value function. The value function can show how the learning differs when learning parameters or discretization is modified. The value function is calculated using Eq. 5.3 introduced in Chapter V.

Figure 42 shows the value functions when the agent follows a fully explorative

policy. The general shapes of each of the value functions are similar. The main difference shown is that the maximum value in the value functions range from 10 to almost 60 as h_{x_i} increases. The reason is that with fewer states to visit, the agent is able to visit and reinforce every state more often than if there are a larger number of states. Also, when there are a larger number of states it takes more visits to each state for the rewards to propagate back into neutral or non-goal states. This results in the sharp decline in value from the area near the goal to the small values for non-goal states farther from the goal. Notice the sharpness of the decline lessens as h_{x_i} increases. The smoothness of the functions is a result of the policy allowing the agent the chance to visit each state an equal number of times. Note that in Figures 42, 43, and 44, C is an abbreviation for Camber, and T is an abbreviation for Thickness.

Figure 43 shows the value functions for a policy annealing from fully explorative to mostly exploitative. Notice the shapes of the value functions now differ between each h_{x_i}. Figure 43.a shows two prominent peaks in the value function. This results from an unequal distribution of state visits early on in the learning. At the outset the agent explores randomly, but given the number of state-action pairs, the agent does not necessarily have time to evenly propagate rewards through the action-value function. For this example of learning, the agent visited the states near the peaks of Figure 43.a early. As the policy changed to require the agent to exploit its knowledge more often, the agent favored the two areas it had already explored the most. This effect resulted in the two peaks. A similar phenomenon occurred for $h_{x_i} = 0.25\%$ and is shown in Figure 43.b. There are a couple small peaks that are not as extreme as those seen in Figure 43.a. This result means that the agent was able to more evenly explore all the states before the policy changed. Given the fewer number of states, this is to be expected. The trend continues as h_{x_i} increases. Figure 42.d is very similar to Figure 43.d. From this figure it is evident that as the number of states decrease,

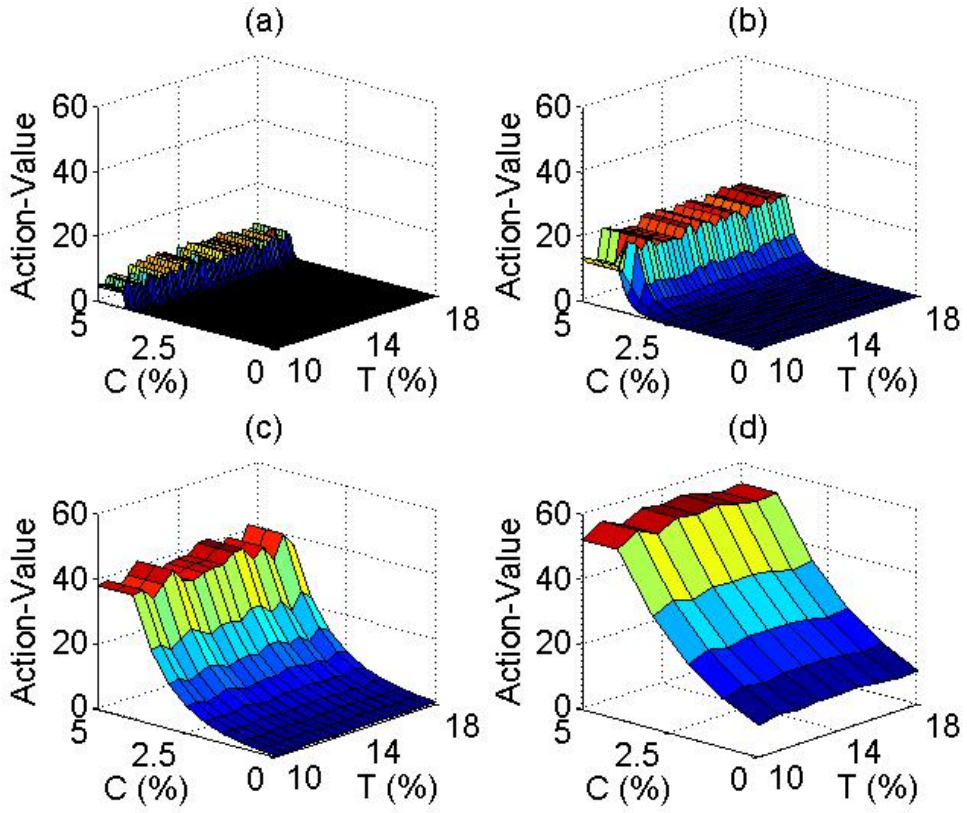

Fig. 42. Value Functions for 100% Exploration: a) $h_{x_i} = 0.10\%$, b) $h_{x_i} = 0.25\%$, c) $h_{x_i} = 0.50\%$, and d) $h_{x_i} = 1.00\%$

the policy the agent follows while learning affects the final action-value function less and less.

Figure 44 shows the value functions resulting from both an annealing policy as well as the discount factor, γ, increasing from 0.0 to 0.7 as learning progresses. Figure 44.a shows a similar prominent peak in the value function as that seen in Figure 43.a. This peak is again a result of the policy followed and the total number of states. A peak is not evident in Figure 44.b as it was in Figure 43.b. One reason is that the discount factor early in the learning is equal to or near zero. When the discount factor

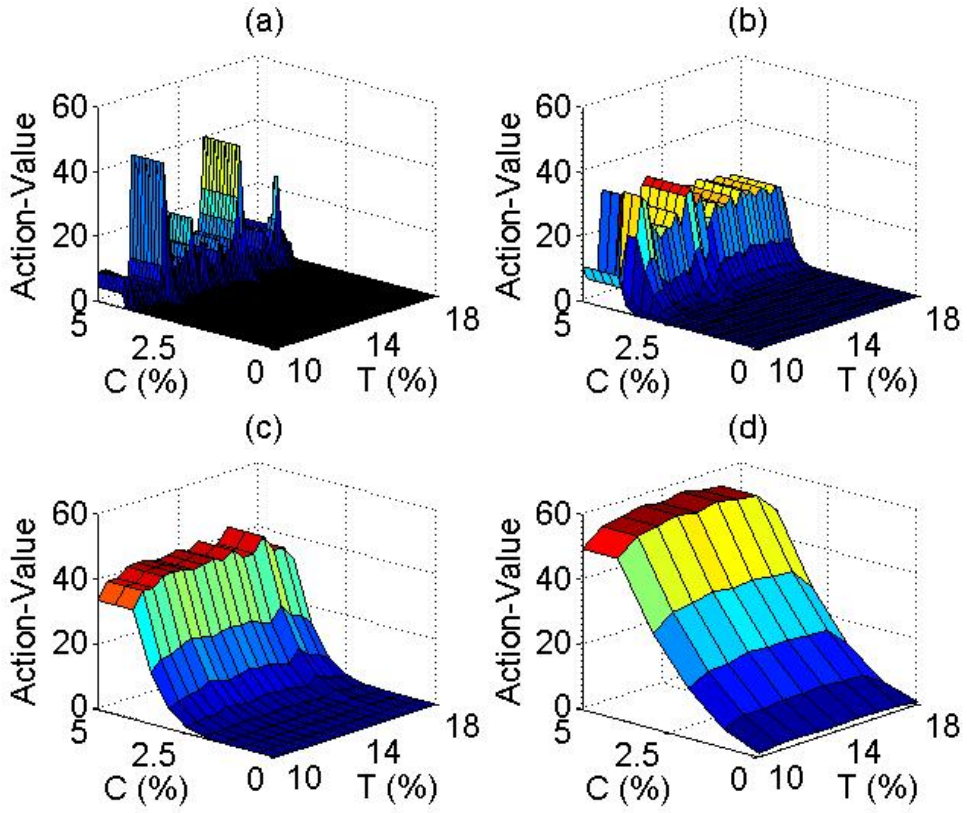

Fig. 43. Value Functions for Annealing of ε: a) $h_{x_i} = 0.10\%$, b) $h_{x_i} = 0.25\%$, c) $h_{x_i} = 0.50\%$, and d) $h_{x_i} = 1.00\%$

is equal to zero, the agent updates the action-value based only on whatever reward it receives. No update in the form of $\gamma \max_{a'} Q(s', a')$ is added to the value function. Another reason is the policy followed as noted previously. Once again as the number of states reduces to that shown in Figure 44.d, the value function is not appreciably different from the value functions in Figures 42.d and 43.d.

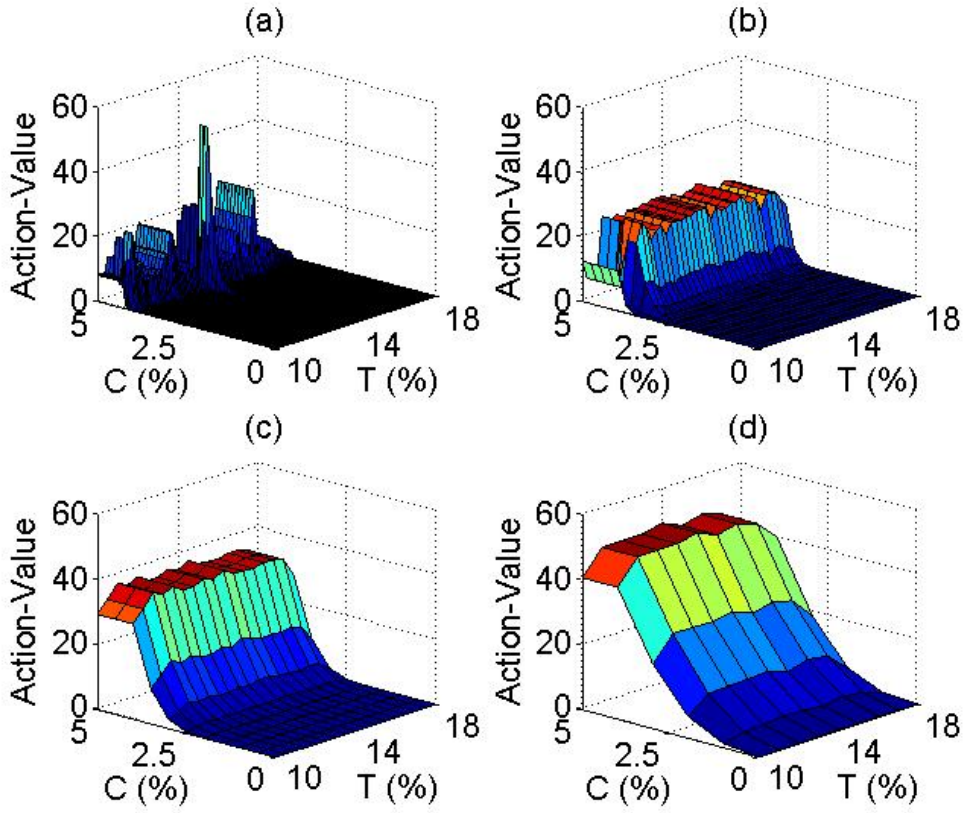

Fig. 44. Value Functions for Annealing of ε and γ: a) $h_{x_i} = 0.10\%$, b) $h_{x_i} = 0.25\%$, c) $h_{x_i} = 0.50\%$, and d) $h_{x_i} = 1.00\%$

c. Developmental Stage 3: Four Morphing Parameter Testing

The purpose of this final developmental stage of the basic morphing airfoil reinforcement learning problem is to demonstrate learning performance of the reinforcement learning agent when integrated with the full four morphing parameter airfoil. The purpose is also to demonstrate the agent commanding smooth transitions from one shape to another through a series of commanded goals. The agent learns the action-value function for four different goals using the h_{x_i} listed in Table XXI. The goals are listed in Table XXII. The range associated with the goal is necessary given the

Table XXI. Distance Between Adjacent Vertices for Morphing Airfoil Developmental Stage 3

h_{x_i}	Value
h_{x_1}	1.00
h_{x_2}	1.00
h_{x_3}	0.10
h_{x_4}	1.00

Table XXII. Learning Goals for Morphing Airfoil Developmental Stage 3

c_l Goal	Range
-0.2	± 0.05
0.0	± 0.05
0.2	± 0.05
≥ 0.4	—

discretization of the state space. A smaller range would constitute a finer discretization of the state space, more learning episodes, and longer computational time. The opposite is true for a coarser discretization. For the morphing airfoil problem, either a balance between state space discretization and goal refinement must be made, a more creative discretization method must be employed, or fully continuous state and/or action space learning must be employed. For the purposes of this developmental stage, the former is used.

For the purpose of direct comparison between the discretized state-space effected by the different h_{x_i} sets, the chord, angle-of-attack, number of episodes, and aerody-

Table XXIII. Learning Parameter Constants for Morphing Airfoil Developmental Stage 3

Parameter	Value
Chord	$1m$
Episodes	10000
α	0.01
γ	0.7

namic goal are the same for each round of learning. The values of these constants are listed in Table XXIII. The reward scheme for each round a learning is defined above by Table XIV.

As before, for each of the 5000 learning episodes, the agent begins in a random initial state that is not classified as a goal state. The agent then explores the state space according to a random policy, $\varepsilon = 0$, until it either hits the predefined limit of total number of actions or finds a goal state. Should the agent encounter a boundary, that boundary location is noted by negative reinforcement, and the agent chooses another action.

Monte Carlo Simulation and Results: The action-value function resulting from each learned goal is analyzed by the Monte Carlo simulation for policy performance every 200 episodes. The simulation is run 1000 times instead of 500 each time the learning is paused to get an accurate measure of the success or failure of the agent and the learning algorithm as the learning is refined. The results for the four learned goals are shown in Figure 45.

These results show about a $92\% - 96\%$ success rate for all the goals the agent

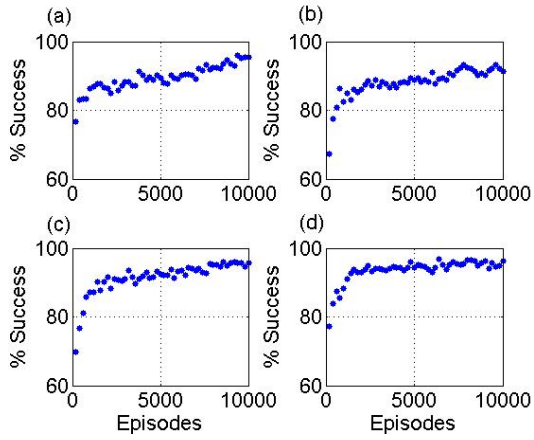

Fig. 45. Agent Learning Success Results: a) $c_l \geq 0.4$, b) $c_l = 0.0 \pm 0.05$, c) $c_l = -0.2 \pm 0.05$, and d) $c_l = 0.2 \pm 0.05$

learned. The convergence rate, defined here as change in success percentage divided by the change in number of episodes, is high early in the learning and decreases to slowly approach 100% success. The exception is Figure 45.a, which appears to be linearly approaching 100% success. After a couple hundred more episodes, this action-value function would also level off to a slow approach to 100%. Also, note that the agent was allowed a 5% probability of choosing a random action. This fact means that some of the failures of the agent to reach the goal can be attributed to the agent choosing a random action and encountering a boundary. Taking these results into consideration, the final action-value functions are deemed sufficiently converged and usable by the agent.

Series of Commanded Goals: This subsection presents examples of the agent's ability to smoothly transition from one shape to another to meet different goals. The environment the agent inhabits is set up such that four goals are commanded in a series. The agent has 200 decision/action steps with which to control and change the

Table XXIV. Goal Series for Morphing Airfoil Developmental Stage 3

Decision Steps	Goal
$0 - 50$	$c_l = 0.0$
$51 - 100$	$c_l = 0.2$
$101 - 150$	$c_l = -0.2$
$151 - 200$	$c_l \geq 0.4$

Table XXV. Initial State of Agent for Examples 1-3 for Morphing Airfoil Developmental Stage 3

State	Example 1	Example 2	Example 3
Thickness (%)	10.0	12.0	15.0
Camber (%)	0.0	3.0	1.0
Max Camber Location	0.4	0.4	0.2
Angle-of-Attack (deg)	0.0	-4.0	-4.0

shape of the airfoil to meet each goal. Each goal is allotted 50 decision steps. This series of goals is analogous to the agent "flying" through a series of flight conditions, such as climb, cruise, dash, or dive. The agent then must quickly change its shape to "fly" well in each flight condition. The goal series the agent must navigate is listed in Table XXIV.

Three representative examples are shown in the following figures. Each example must meet the goals listed in Table XXIV. Each example has the agent start from a different initial state. These initial states are listed in Table XXV.

Figures 46 and 47 illustrate the results of the these three examples. Figure 46

shows the upper and lower bounds of the lift coefficient goals the agent must strive to meet. The examples show that the changes in lift coefficient resulting from agent commanded changes in shape fall within the bounds of all four goals after only a few decision steps. The agent marches directly toward its goal in each example. This directness is desired given the simplicity of the reward structure employed in the learning presented in this paper. The rewards can be replaced by a more complex cost function or by reward shaping, which could result in a less direct, though still good, path toward each goal. The fact that the agent commands shape changes that drives the lift coefficient directly toward the goals means that the agent achieved good convergence of the action-value function for this problem.

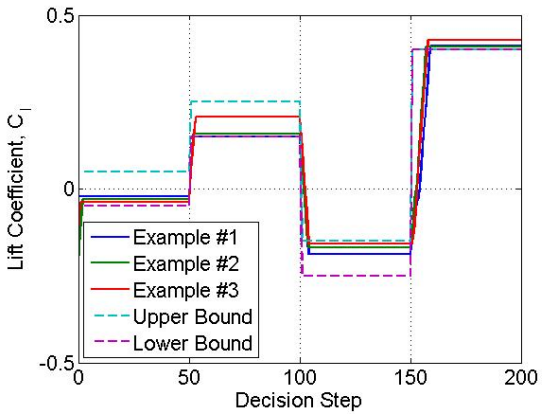

Fig. 46. Airfoil Lift Coefficient Results for Morphing Airfoil Developmental Stage 3

Figure 47 depicts the airfoil states at each decision step. All three examples show that the agent mainly utilizes commanded changes in camber and angle-of-attack to manipulate airfoil lift coefficient. The agent only sparingly commands changes in thickness and max camber location. Changes in angle-of-attack are commanded at the beginning of transitions between goals. These changes result in relatively

large changes in lift coefficient. As the agent approaches the proximity of the goal, commanded changes in camber are favored. The reason is that a change in camber results in a smaller change in lift coefficient. This flexibility allows the agent to approach and achieve the goal without overshooting it.

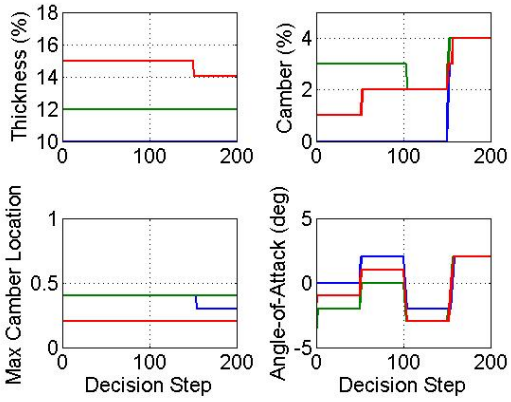

Fig. 47. Airfoil Morphing Parameter Results for Morphing Airfoil Developmental Stage 3

2. Case 2: Q-Learning with AAG

AAG is now added to Q-learning to test on the morphing airfoil. The general setup of the problem is much the same as before except that a few extra parameters must be defined for AAG. Each finer discretization is determined using a preset factor, g_f, applied to the coarser discretization, such that $h_{x_i}^{j+1} = g_f h_{x_i}^{j}$. The number of levels of discretization or resolution as defined earlier is M. The parameters for this problem are listed in Table XXVI.

As for the initial application of Q-learning described above, there were several stages during the initial development and application of AAG. They entail how the action-value matrix is carried over from one level of discretization to the next as well

Table XXVI. Airfoil AAG Parameters

Parameter	Value
$h^1_{x_1}$	0.50
$h^1_{x_2}$	0.50
g_f	0.20
M	3
Goal, g	$c_l = 0.3$
Initial Range, g_{r_1}	0.025

as the reward scheme for the problem and are described in more detail in the next 3 subsections.

a. Developmental Stage 1

The first stage is designated as the "brute force" method. The original motivation for AAG was to separate a problem into a series of smaller problems with successively more refined discretization and goal range focused in the Region Of Interest. In line with this mentality, the action-value function is reinitialized for each level of discretization, essentially beginning each with a blank slate and thus making each level of discretization a truly separate reinforcement learning problem and patching the action-value functions together after learning is complete. Initial tests allowed the agent to explore the state-space over 5000 episodes for each level of discretization, but did not yield good results. Therefore, in this brute force stage, the agent is allowed 30000 episodes for each level of discretization. The reward scheme is that shown in Table XIV. The reasoning behind this is to determine if convergence can be achieved in a reasonable number of episodes using this approach, or if some other approach

Table XXVII. Morphing Airfoil Policy Color Scheme

Action	Color
$-h_{x_1}$	Blue
$+h_{x_1}$	Cyan
$-h_{x_2}$	Yellow
$+h_{x_2}$	Red

is need. Figure 48 shows the value function and greedy policy calculated using Eqs. 5.1 and 5.3 from the final action-value function. The actions for the policy are color coded and are listed in Table XXVII.

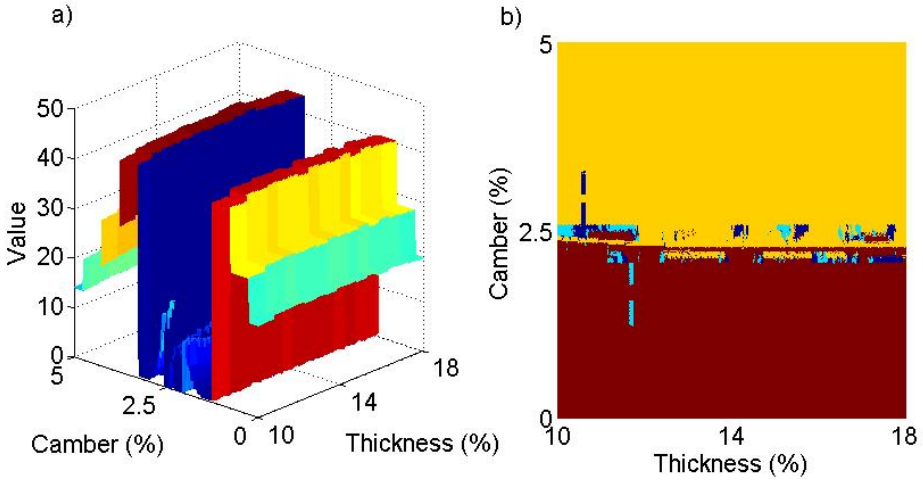

Fig. 48. AAG Airfoil Developmental Stage 1: Value Function (a) and Policy Representation (b)

Both images show that the Region Of Interest for this case is a line that bisects the state at a nearly constant value of camber, ˜2.4%. The three levels of discretization are easy to see in value function. The contours near the camber bounds of the

state-space are blocky in nature whereas the contours near the center of camber axis, and thus near the lift coefficient goal, are more refined. The sharp descent to 0 of the value function along the camber axis markedly illustrates the reinitialization of the action-value function for each level of discretization. Also, the peaks in each finer discretization are not as large as for the initial discretization. This is due to the larger number of states at these finer resolutions. The action-values have not had as much opportunity to propagate back.

The policy representation indicates that despite the 90000 total episodes, the areas of finer discretization in the Region Of Interest around the goal did not converge to an acceptable policy. The splotches of abutting colors indicating opposing actions are areas in which the agent, when fully exploiting this policy, will simply move back and forth between a couple states and never make progress toward the goal. In essence, the agent is stuck. This fact means that this brute force method with a traditional reward structure is not a good approach for the application of AAG.

b. Developmental Stage 2

This stage takes a different approach in how the action-value function is handled from one level of discretization to the next. Since each new discretization is in a Region Of Interest that is a part of the original problem and not totally separate, it seems logical that knowledge stored in action-value function should be retained and refined. This means that the learning is refined in the Region Of Interest rather than starting with nothing. The agent is allowed 5000 episodes for each level of discretization, and the reward structure in Table XIV is used. Figure 49 shows the value function and policy for this developmental stage.

The value function shows a different structure than that for the first developmental stage. There is an evident increasing step structure from the camber bounds

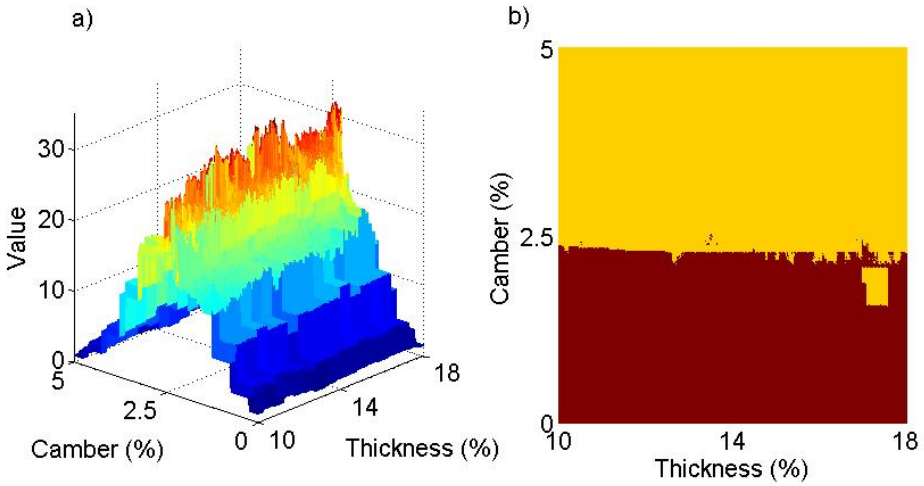

Fig. 49. AAG Airfoil Developmental Stage 2: Value Function (a) and Policy Representation (b)

to a center peak that runs lengthwise to the thickness axis. The lack of drops in the value function back to 0 illustrates the retention of knowledge in the action-value function. The central peak, at a near constant camber of ˜2.4%, is where the agent received the most reinforcement and is where the goal is located. The blocky nature of the initial discretization and the more refined, though still blocky, nature of the 2^{nd} and 3^{rd} discretizations is more evident in this value-function. The jagged edge of the peak has finer detail than the outer edges of the contour near the outer camber bounds.

The policy for this stage is more promising than for the first stage. For the majority of the state-space, the policy has the agent move to states that meet the lift coefficient goal. There are only a few small areas and one large area in which the color coded actions show that the agent would get stuck and be unable to progress to a goal state. This is a promising step and signifies that retaining knowledge in the action-value function from one level to the next should be incorporated.

Table XXVIII. States and State-Action Pairs for Morphing Airfoil AAG Developmental Stage 3

	States	State-Action Pairs
Multi-Resolution	8881	35524
Single-Resolution	100651	402604

c. Developmental Stage 3

The successful and final stage retains knowledge of the action-value function as the previous stage, but uses a different reward scheme. For this stage the reward is changed to that described the Eq. 8.11. The agent is again allowed 5000 episodes with which to explore the state-space for each level of discretization. Since this approach is the first truly successful application of AAG, the learning performance is analyzed in several ways. First the dimensionality of the multi-resolution action-value function is compared with that of the full state-space discretized at the finest level. Next Monte Carlo simulation performance results, as described in Chapter V, are analyzed. Finally, the final value function and policy are considered.

Dimensionality: The number of states and state-action pairs for both the actual learned multi-resolution problem and the hypothetical single-resolution problem described by Eqs. 3.11 and 3.12 are shown in Table XXVIII. The multi-resolution method reduces the number of state-action pairs the agent must visit by an entire order of magnitude. This greatly simplifies the learning problem and encourages faster convergence because the agent has the opportunity to visit each state more often in the allowed number of episodes.

Monte Carlo Simulation: Figure 50 shows the results of the Monte Carlo simulation performance analysis. Each set of 5000 episodes is for a different level of discretization, from coarsest to finest. The final star at 15000 episodes is for the simulation using the full power of the AAG method. The final range for the goal in this problem is 0.001. This figure shows that the first and second levels of discretization converge quickly, within 400 episodes. The third level takes about 1200 episodes to converge above a 98% success rate and continues to improve as learning continues. The final simulation shows that the agent can use its final set of information to reach the goal with over a 99% success rate.

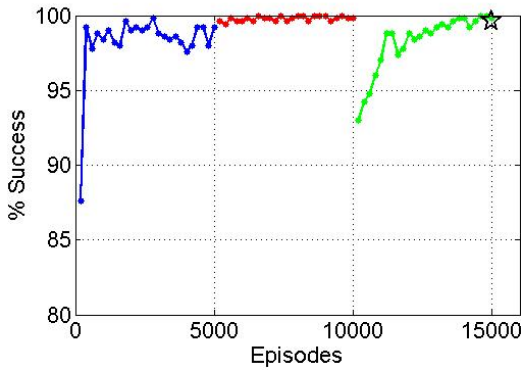

Fig. 50. AAG Airfoil Developmental Stage 3: Monte Carlo Simulation Results

Value Function and Policy Analysis: Figure 51 shows the final value function and greedy policy. Toward the upper and lower extremes of the camber axis, the function is very blocky as shown in the earlier stages. The sudden increase in value function as the camber approaches 2.4% is a result of learning on the second level of discretization. Notice the function is less blocky in this region. The minimal ridge in the middle of the function is effectively the goal of $c_l = 0.3$ and where the reward function approaches 0. This minimal ridge is reflected in the graphic of the greedy policy. This figure shows

that change in camber has the greatest influence in affecting airfoil lift coefficient, enough such that thickness may not be a necessary morphing parameter in future iterations of this problem. There are also no areas in the policy representation that prevents that agent from progressing smoothly to a goal state. This figure and the other analyses show that this approach is successful and should be the starting point for all subsequent applications of AAG.

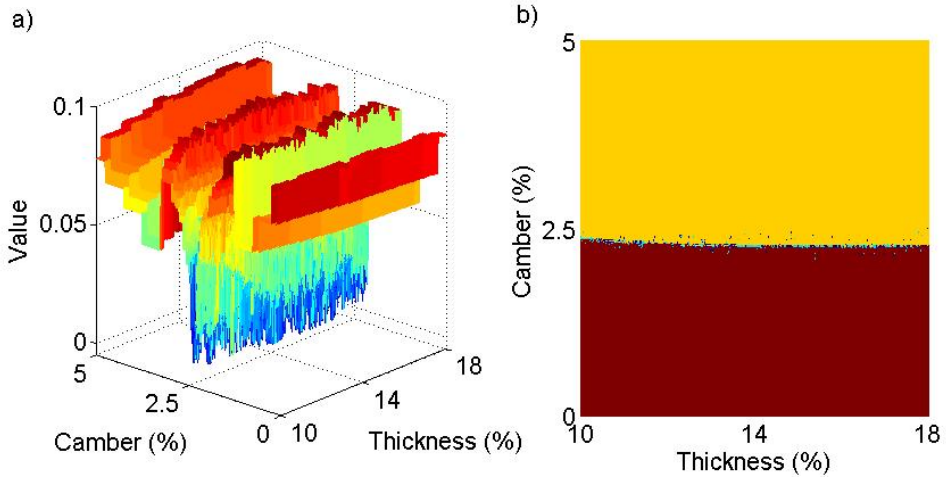

Fig. 51. AAG Airfoil Developmental Stage 3: Value Function (a) and Policy Representation (b)

3. Case 3: Q-Learning with AAG and PC

Now that AAG has been applied and tested with success, the policy comparison and policy performance stopping criteria are added to the Q-learning with AAG algorithm and tested on the morphing airfoil reinforcement learning problem. The problem for this example is set up similarly to that of Case 2. The axes and AAG parameters are listed above in Table XXVI. Knowledge of the action-value function is carried over between discretizations and the reward structure defined by Eq. 8.11 is used.

Table XXIX. States and State-Action Pairs for Morphing Airfoil with AAG and PC

	States	State-Action Pairs
Multi-Resolution	8129	32516
Single-Resolution	100651	402604

The learning is analyzed in the same manner as was the third developmental stage above. The dimensionality of the multi-resolution action-value function is compared with that of the full state-space discretized at the finest level, the Monte Carlo simulation performance results are analyzed, and the final value function and policy are considered.

a. Dimensionality

The dimensionality for this problem is listed in Table XXIX. The multi-resolution method reduces the number of state-action pairs the agent must visit by an entire order of magnitude. Though this problem is essentially the same as the previous case, there are about 800 fewer states in the action-value function. This is due to a more refined method of tracking the edge of the Region Of Interest than the block structure first described in Eq. 3.11. This method simply records the outer bounds more accurately. The difference in total states, however, is small, so Eq. 3.11 still applies to the general understanding of the problem.

b. Monte Carlo Simulation

Figure 52 shows the results of the Monte Carlo simulation performance analysis for this problem. Each level of discretization is allowed a possible 5000 episodes. The final range for the goal in this problem is 0.001. Recall that learning on each level of

discretization is terminated when there is less than 5% change in the policy extracted from the action-value function and the policy performance analysis yields greater than 98% success.

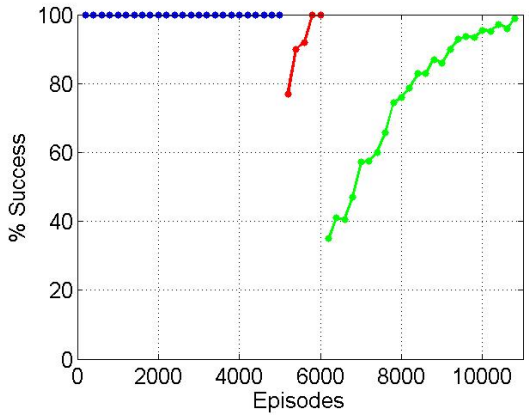

Fig. 52. Airfoil with AAG and PC: Monte Carlo Simulation Results

The figure shows that the first discretization reaches 100% within 200 episodes, yet the learning is not terminated until the 5000 allowed episodes is exhausted. This indicates that the policy itself is changing enough such that there is not at least 95% similarity between one extracted policy and the next. There are 187 states in the coarsest discretization. If just 10 of these change every 200 episodes, then the policy comparison criterion is not met. The high performance measure indicates that the policy is "good", so we can conclude that the changing states are within the Region Of Interest itself.

The second discretization converges and reaches 100% success within 1000 episodes. The third discretization reaches $>$ 98% success within 4800 episodes. The learning thus only ran for 10800 episodes, 4200 episodes less than the total allowed number of 15000. The policy comparison and performance criteria reduced the total episodes

by 28% with respect to the total allowed number of episodes. This number must be taken in context. The 15000 is based on the prior tests using 5000 episodes for a given discretization level. Obviously, if 7500 episodes per discretization level is chosen, then the reduction in total episodes is larger.

c. Value Function and Policy Analysis

Figure 53 shows the final value function and greedy policy for this problem. The general topography of the value function is similar to both shown in Figures 49 and 51. The most activity is again near a camber of 2.4%, but the valleys of the surface are less extreme than that seen in Figure 51. This is due to fewer number of episodes in the second and third discretizations. The learning had less time to mature, in a sense. It is hoped that this "less extreme" action-value function will be more easily approximated. The greedy policy again shows similarities to Figure 51. However, there are more small patches of blue and cyan that are not evident in Figure 51. These are areas that were not learned quite as well as in the previous example. From the figures shown in this case, though, this approach is considered successful.

4. Case 4: Q-Learning with MGAP

The final step is to add the genetic algorithm for function approximation to the Q-learning algorithm with AAG and PC and demonstrate it with the morphing airfoil reinforcement learning problem. This final component completes the Q-learning with MGAP algorithm. The problem for this example is set up similarly to Case 2 and Case 3. The axes and AAG parameters are again listed in Table XXVI. Knowledge of the action-value function is carried over between discretizations and the reward structure defined by Eq. 8.11 is used. When learning is paused, the action-value function is approximated using the genetic algorithm developed in Chapter IV. The

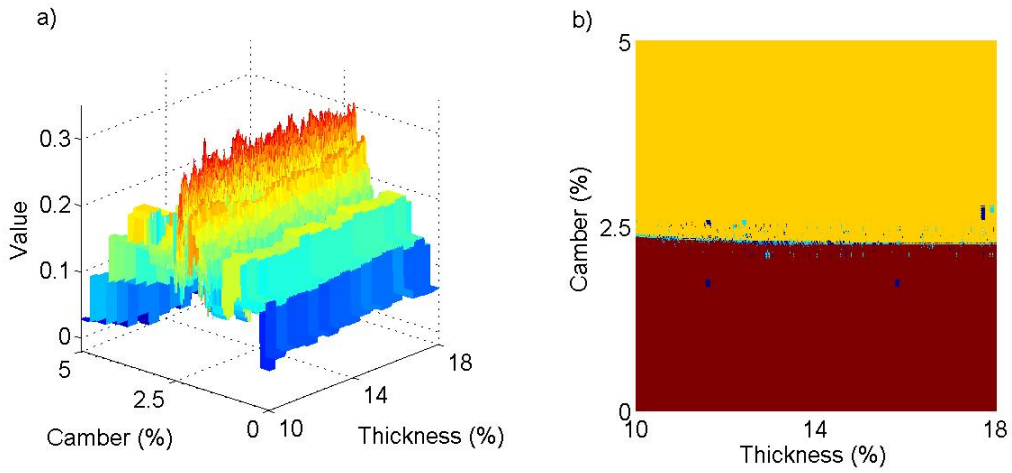

Fig. 53. Airfoil with AAG and PC: Value Function (a) and Policy Representation (b)

PC stopping criteria developed in Chapter V are applied to this approximation. The learning with the full algorithm is analyzed in a similar manner as before. The final set of basis functions for each level of discretization are presented, the Monte Carlo simulation performance results are analyzed, and the final approximated value function and associated policy are considered.

a. Approximation

The GA determines the set of basis functions and degree or number of knots for those basis functions each time the learning is paused, in this case every 200 episodes. It is this approximation that is used for the policy comparison and performance analysis. In a sense the selections made by the GA evolves as the action-value function in tabular form evolves. As mentioned in previous chapters, each $Q(s, a_i)$ has a table of the learned action-values for that action, a_i. The GA is applied to each of these separately to yield $\tilde{Q}(s, a_i)$. Since there are four actions for this problem, there are four approximations that must be made to fully approximate the action-value

Table XXX. Approximation Bit Strings for Each Level of Discretization for Morphing Airfoil

	Level 1	Level 2	Level 3
$\tilde{Q}(s, a_1)$	1001111	1001111	1011110
$\tilde{Q}(s, a_2)$	1001111	1001111	1011111
$\tilde{Q}(s, a_3)$	1001111	1011111	1011110
$\tilde{Q}(s, a_4)$	1001111	1011111	1011111

function for this problem. Listed in Table XXX are the bit strings encoding the approximations after the final episode of each level of discretization.

The basis functions for the end of the first level of discretization show consistency between of each of the actions. Each $\tilde{Q}(s, a_i)$ for this level is best approximated, according to the GA, by linear RBFs with $(15+2)(15+2) = 17^2 = 289$ evenly distributed knots or centers. The approximations for the end of the second level of discretization is similar to the first. The first two actions are again approximated by linear RBFs with 289 centers, while the other two actions are best approximated by cubic RBFs with 289 centers. This continuity indicates that the major features of the action-value function changed only slightly from one level of discretization to the next. The approximations for the final action-value function are again very similar to the first two. The GA states that all four $\tilde{Q}(s, a_i)$ are best approximated by cubic RBFs, though $\tilde{Q}(s, a_1)$ and $\tilde{Q}(s, a_3)$ only need 256 centers as opposed to 289 centers for $\tilde{Q}(s, a_2)$ and $\tilde{Q}(s, a_4)$. These numbers are summarized in Table XXXI.

Consider the final approximations as compared to the tabulated action-value function. There are 8129 states and thus 32516 state-action pairs with associated preferences or action-values in the discretized action-value function. Implementing a

Table XXXI. Comparison of Data for Tabulated and Approximated Action-Value Function for Morphing Airfoil

	Level 1 Discretization		Level 2 Discretization		Level 3 Discretization		
	Centers	Weights	Centers	Weights	Centers	Weights	States
$Q(s, a_1)$	289	289	289	289	256	256	8129
$Q(s, a_2)$	289	289	289	289	289	289	8129
$Q(s, a_3)$	289	289	289	289	256	256	8129
$Q(s, a_4)$	289	289	289	289	289	289	8129
Totals	289	1156	289	1156	545	1090	32516

working controller using this learned discretized data entails carrying around large tables and conducting table look-ups and interpolation whenever the data is needed. The approximations are much more compact. If only the final approximation is needed, then only the locations of the centers and their associated weights for each $\tilde{Q}(s, a_1)$ for that approximation must be stored. This means $256 + 289 = 545$ center locations (do not need duplicates) and $256 + 289 + 256 + 289 = 1090$ weights are needed. That is a total of 1635 numbers that must be stored and used, which is almost 95% less than that for the discretized action-value function.

It is also possible that the approximations made at the end of learning of the first and second levels of discretization are needed to implement a working controller. This might be necessary should the final approximation only accurately capture the function in and around the final area of interest. With the higher concentration of states in that area, the approximation may be forced to accurately capture that local behavior and lose accuracy in the global behavior that was accurately captured during

learning and approximation of the coarser discretizations. If that is the case the agent decides to use the approximation from the first and second levels of discretization when it is far away from the goal and only moderately close to the goal. It then uses the final approximation when in close proximity to the goal, namely in the region learned using the finest discretization. Therefore, the agent retains information for all three approximations. This means that $256 + 289 = 545$ center locations are needed, and $1156 + 1156 + 1090 = 3402$ weights are needed. That is a total of 3947 numbers maximum that must be stored and used, which is still 87% less than that for the discretized action-value function.

b. Monte Carlo Simulation

Figure 54 shows the results of the Monte Carlo simulation performance analysis for this problem using the approximated action-value function every 200 episodes. Each level of discretization is again allowed a possible 5000 episodes. The final range for the goal in this problem is 0.001. Learning on each level of discretization for this case is terminated when there is less than 5% change in the policy extracted from the approximated action-value function in Eq. 5.2 and the policy performance analysis yields greater than 98% success.

The figure shows that the first discretization reaches 100% within 200 episodes, and the learning is terminated after 2400 episodes. Each approximated action-value function in the first discretization has 100% success. The explanation for this occurrence is the same as for the previous case. The second discretization converges and reaches 100% success within 1200 episodes. The third discretization reaches 96% success by the end of the 5000 episodes. The trend indicates that if learning were allowed to continue, then $>$ 98% success would be reached in another 200 to 600 episodes. Based on the learned data here, though, only 8600 episodes were needed of

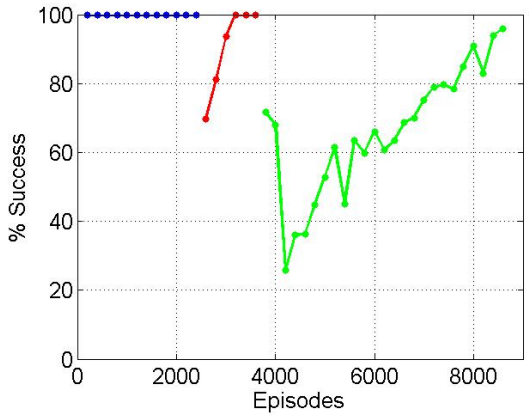

Fig. 54. Airfoil with MGAP: Monte Carlo Simulation Results

the 15000 allowed episodes, which is a 42.6% reduction.

c. Value Function and Policy Analysis

The following figures show the value functions and greedy policies based on the approximated action-value functions for each level of discretization. These are determined using Eqs. 5.2 and 5.4. It is informative to consider each one individually to see how well the approximation captures or does not capture the global and local (i.e. in and around the Region Of Interest) behavior of the action-value function.

Figure 55 illustrates the value function and greedy policy based on the approximated action-value function after the final episode learned on the coarsest discretization, episode 2400. The value function suggests that the action-value function is a fairly simple function to approximate. The easy rise to the crest at a camber of 2.5% is a large global feature that the least-squares approximation easily approximates. This is reflected in the Monte Carlo simulation as well as the greedy policy. The policy is consistent with the previous cases and is exactly what is expected for the

coarsest discretization. The policy guides the agent to the region containing the goal of $c_l = 0.3 \pm 0.025$, which is $10.0\% \leq x_1 \leq 18.0\%$ and $2.0\% \leq x_2 \leq 3.0\%$. Within that region the color coded actions guide the agent to the those states that are closest to $c_l = 0.3$.

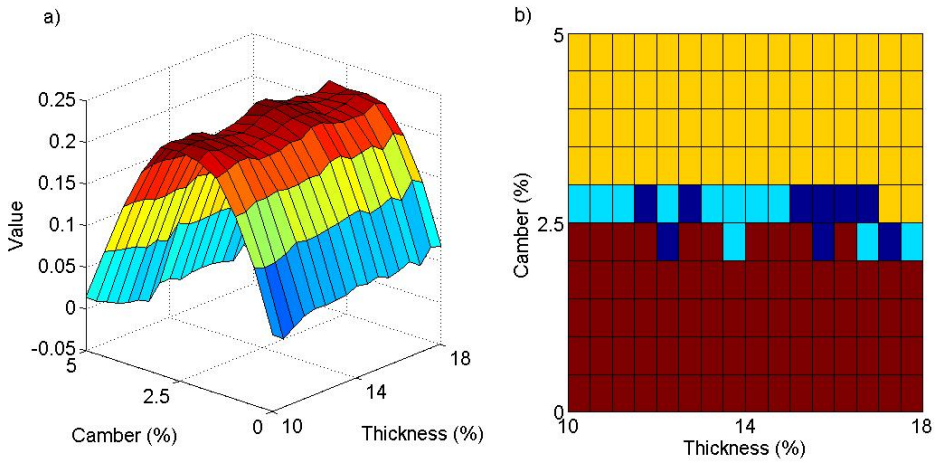

Fig. 55. Airfoil with MGAP, Episode 2400: Approximate Value Function (a) and Policy Representation (b)

Figure 56 shows the value function and greedy policy based on the approximated action-value function after the final episode learned on the second discretization, episode 3600. The value function is consistent with what is expected after the second discretization based on the previous examples. There are some oscillations along the crest that have not been seen before that are a result of the approximation. The visualization of the policy, however, indicates that these oscillations do not overly affect it, as is supported by the policy performance analysis above. The policy after 3600 episodes is consistent with the policy after 2200 episodes, and it shows that the additional learning filled in the detail around the more specific Region Of Interest, namely $c_l = 0.3 \pm 0.005$, which is $10.0\% \leq x_1 \leq 18.0\%$ and $2.1\% \leq x_2 \leq 2.5\%$.

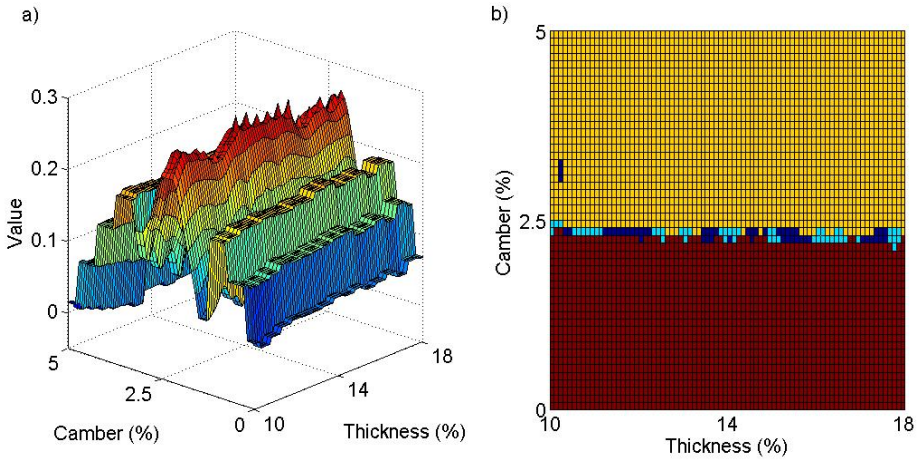

Fig. 56. Airfoil with MGAP, Episode 3600: Approximate Value Function (a) and Policy Representation (b)

Figure 57 shows the final value function and greedy policy based on the approximated action-value function for this problem. The general topography of the value function is similar to the previous case using AAG and PC only. The most activity is again near a camber of 2.4%, but it has a smooth and oscillatory characteristic. This is evidence of the approximation. The greedy policy shows some marked differences from previous cases and inconsistency with the policies after 2200 and 3600 episodes in the form of the large blue and cyan patches. According to the Monte Carlo simulation the policy for the finest discretization in the region defined by $10.0\% \leq x_1 \leq 18.0\%$ and $2.1\% \leq x_2 \leq 2.5\%$ does successfully guide the agent to the final Region Of Interest defined by $c_l = 0.3 \pm 0.001$. However, some of the global behavior is lost beyond this region of fine discretization. Since the approximation after 3600 episodes captures the "good" behavior beyond these bounds, it can be used by the agent in those regions and the approximation after 8600 episodes can be used in the region close to the goal.

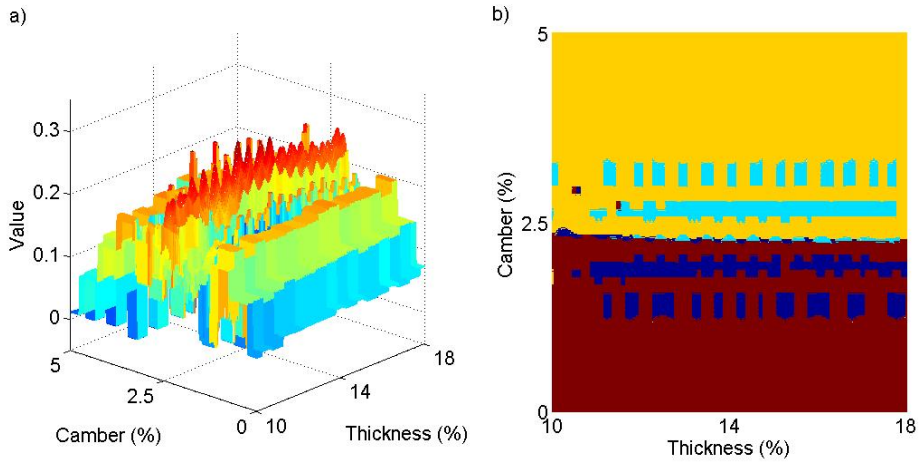

Fig. 57. Airfoil with MGAP, Episode 8600: Approximate Value Function (a) and Policy Representation (b)

For comparison, Figure 58 shows the final value function and greedy policy determined by Eqs. 5.3 and 5.1 based on the tabular action-value function. The approximations after 2200 and 3600 episodes are consistent with the raw data in this figure. The final approximation shows good agreement with the region around a camber of 2.4%, but is not consistent with the raw data beyond that region.

D. Summary

In this chapter the airfoil was cast as a reinforcement learning problem and run through the gamut of various components of the algorithm developed in this research. Results show that the agent learned a variety of goals for an airfoil with both two and four state variables using Q-learning. The agent then used the learned data autonomously to meet a series of goals in simulation to within ± 0.05 of the aerodynamic goal. Results also successfully demonstrate Q-learning with multi-resolution state-space discretization. Initial attempts in which the agent had 90000 episodes

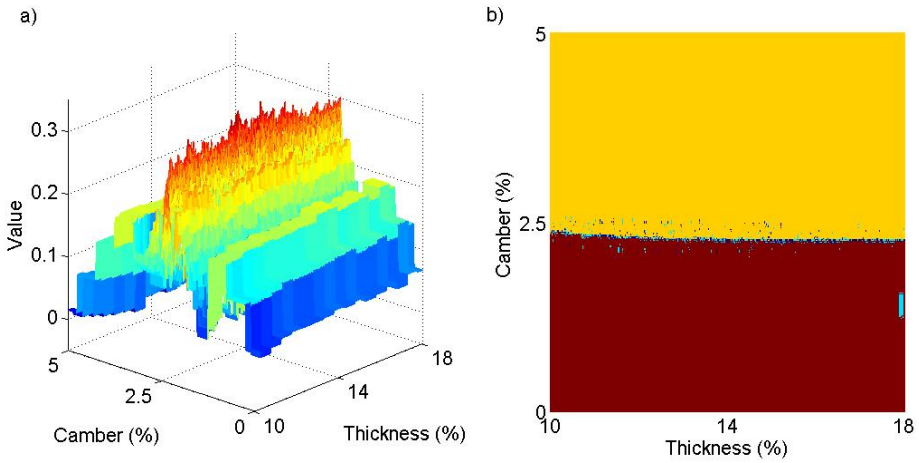

Fig. 58. Airfoil with MGAP, Episode 8600: Value Function (a) and Policy Representation (b)

to explore the state-space proved unsuccessful. Introducing a gradient based reward function allowed the agent to learn the goal of $c_l = 0.3 \pm 0.001$ in 15000 episodes. Incorporating multi-resolution state-space discretization and policy comparison into the same problem enabled the agent to learn the same goal, $c_l = 0.3 \pm 0.001$, in only 10800 episodes, which is a 28% reduction. The final set of results demonstrate Q-learning with MGAP in which the policy extracted from the approximated action-value function converged to achieve a goal of $c_l = 0.3 \pm 0.001$ in 8600 episodes, which is a 42.6% reduction from the possible 15000 episodes. The final approximation did not accurately capture the global behavior of the action-value function, so it can not be used when the agent is far from the goal. Instead the agent uses the approximations at the end of the first or second level of discretiation with far from the final region of interest and final approximation when near the region.

CHAPTER IX

RECONFIGURABLE SYSTEM EXAMPLE - MORPHING WING

The second application of this algorithm to a reconfigurable system is a morphing wing (Figure 59). Like the airfoil, rather than finding a couple optimal shapes that meet some criteria and a separate optimal controller to maneuver from one shape to the other, the wing is cast as a reinforcement learning problem in which the full spectrum of shapes that meet the flight phase criteria are learned as well as the local transitions that guide the shape from *any* initial shape to a shape that meets the requirements.

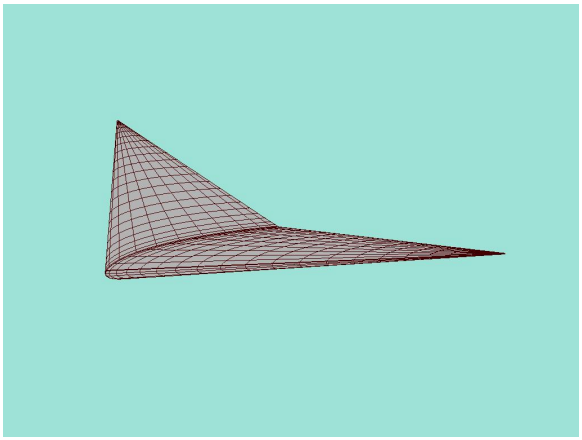

Fig. 59. Representative Wing

Section IX.A discusses the wing model to be cast as the environment of the reinforcement learning problem. Then the full reconfigurable wing reinforcement learning problem is described followed by a series of examples that exercise the many aspects of the problem. The numerical results are relatively brief as compared to airfoil because the major troubleshooting has already been accomplished, and we simply need to apply the algorithm to the reinforcement learning problem.

A. Wing Model

The airfoil is modeled by a CFD code using a constant strength doublet-source panel method. This model calculates the aerodynamic properties of a wing given a set of 13 inputs:

- Wing root airfoil thickness
- Wing tip airfoil thickness
- Wing root airfoil camber
- Wing tip airfoil camber
- Wing root location of maximum camber
- Wing tip location of maximum camber
- Taper ratio
- Aspect ratio
- Leading edge sweep angle
- Dihedral angle
- Wing span
- Wing twist
- Wing angle-of-attack

Setting these 13 parameters as inputs to the model allow for rapid and efficient calculation of the aerodynamic properties of many different wing configurations or as a single wing changes shape.

A number of assumptions were made during the development of the model. These assumptions are

1. Flow is incompressible.

2. Flow is inviscid.

These assumptions allow for a model that is valid for the linear range of angle-of-attack and sufficient for the purposes of this research.

Calculating the aerodynamic properties of the wing in question begins with considering the basic equation of potential flow theory.

$$\Delta^2\Phi = 0 \tag{9.1}$$

From here commences an involved derivation to calculate the doublet strengths for the panels that lie along the wing, which is beyond the scope of this research. With these strengths the local velocities is calculated, and from there the total velocity, Q_k for each panel is calculated.

Using these total velocities, the pressure coefficient at each panel is calculated using a modified form of Bernoulli's Equation:

$$C_{p_k} = 1 - \frac{Q_k^2}{Q_\infty^2} \tag{9.2}$$

With these pressure coefficients, the non-dimensional aerodynamic forces are found for each panel. This calculation leads to the total aerodynamic forces, which can be determined by summing the contributions from each panel.

$$\Delta C_{F_k} = -\frac{C_{p_k}\Delta S}{S} \cdot \mathbf{n}_k \tag{9.3}$$

A complete development of the wing model used for this research can be found in Reference [184].

B. Wing Cast as a Reinforcement Learning Problem

Casting the morphing wing as a reinforcement learning problem is much the same as it was for the airfoil in Chapter VIII. In the case of the wing, there are no dynamics involved in the shape change. Just as for the airfoil, each commanded change in wing shape yields an immediate response. This formulation effectively removes a potentially complicated aspect of the morphing wing problem from this reinforcement problem and allows for better focus on the actual choice in shape.

The choice of state variables is dependent on two concerns. The state variables must be chosen such that the aerodynamic properties of the airfoil are adequately exploited and the CFD model, which constitutes the *environment*, does not take overly long to calculate the aerodynamic properties of the current wing configuration. There are 13 possible interdependent parameters, and the runtime for the CFD model to calculate one set of forces is on the order of 100 seconds. Running a learning algorithm with this as the environment is not practical, therefore a table of the aerodynamic forces for a subset of the morphing parameters is generated from the CFD model and used as the environment instead. This subset includes leading edge sweep angle, wing span, root chord, and tip chord. The others are held constant.

The agent interacts with its environment in the same manner as the airfoil. The definitions of the x_i axes for all of the examples are defined in Table XXXII.

The *goal*, g, of the agent for this problem is defined by the aerodynamics of the wing. As for the airfoil reinforcement learning problem, every goal has a range, g_r, associated with it. Those vertices whose state yield aerodynamic coefficients, c, from the CFD model that lie in the goal range defined by $g - g_r \leq c \leq g + g_r$ form the pseudogoals of the problem.

The *reward* for this problem is the gradient based reward function that is shown

Table XXXII. Morphing Wing Axis Definitions

x_i	Definition
x_1	Wing Span
x_2	Root Chord
x_3	Tip Chord
x_4	Leading Edge (LE) Sweep Angle

Table XXXIII. Morphing Wing Parameter Limits

Limit	Lower	Upper
Wing Span (m)	5	12
Root Chord (m)	2	4
Tip Chord (m)	0.5	2
LE Sweep Angle (deg)	0	25

again in Eq. 9.4 for ease of reference. The limits for the state variables that define the perimeter of the environment are listed in Table XXXIII.

$$r = |g - c_{n-1}| - |g - c_n| \tag{9.4}$$

C. Numerical Results

The purpose of this numerical example is to demonstrate the learning performance of the reinforcement learning agent utilizing the multi-resolution state-space discretization method, policy comparison and performance stopping criteria, and MGAP on

Table XXXIV. Wing Learning Parameters

Parameter	Value
Episodes	5000
α	0.01
γ	0.7
g	$\left(\frac{C_L}{C_D}\right)_{\max}$
g_{r_1}	6
g_f	0.5
M	3
$\left(h^1_{x_1}, h^1_{x_2}, h^1_{x_3}, h^1_{x_4}\right)$	$(1.0\text{m}, 0.5\text{m}, 0.5\text{m}, 5\,\text{deg})$

a more complex reconfigurable system. It is the next step in developing how the algorithm is applied to the class of highly reconfigurable systems. In each of the two cases presented here, the agent is given the task of learning to maneuver through the state-space to a goal state that satisfies some aerodynamic requirement. The agent is allowed a possible 5000 episodes with which to learn the shape parameter state-space for each level of discretization. Section 1 has the agent learn using both AAG and PC. A section for the agent learning using just PC was not included because it did not add or enhance the presentation of the algorithm in any way. Section 2 presents the second case in which the GA and PC are applied to the morphing wing problem.

Both cases have the same admissible actions, discretization factor g_f, and number of levels of discretization M. These parameters are listed in Table XXXIV. The goal, g, and initial range, g_{r_1}, are the same for both cases as well.

A case describing the results for the full MGAP algorithm applied to this problem is not included for one reason: the computational demands of the genetic algorithm

for this larger problem were such that the supporting computer runs out of memory during the calculation of ω. Therefore, the case with the GA and PC is shown to illustrate show the success of these components. Recommendations are made in the final chapters to alleviate this computational issue in the future.

1. Case 1: Q-Learning with AAG and PC

The first morphing wing case tests Q-learning with AAG and PC. Since there are three levels of discretization, the final range for the goal in this problem is $g_{r_3} = 1.5$. The learning is paused and PC applied every 200 episodes. Analysis of the learning is conducted in a similar manner to many of the airfoil examples. The dimensionality of the multi-resolution action-value function is compared with that of the full state-space discretized at the finest level, and the Monte Carlo simulation performance results are analyzed. The value function and policy are not presented in image form as was the case for the airfoil because of the 5-dimensional nature of those surfaces for this problem.

a. Dimensionality

The dimensionality for this problem is listed in Table XXXV. The multi-resolution method reduces the number of state-action pairs the agent must visit by two orders of magnitude. The number of states is relatively small due to the large initial discretization of the state-space. As such, AAG allows the agent to focus its attention on the small area around the goal rather than the area outside of the Region Of Interest.

b. Monte Carlo Simulation

Figure 60 shows the results of the Monte Carlo simulation performance analysis for this problem. Each level of discretization is allowed a possible 5000 episodes. Recall

Table XXXV. States and State-Action Pairs for Morphing Wing Case 1

	States	State-Action Pairs
Multi-Resolution	927	7416
Single-Resolution	69615	556920

that learning on each level of discretization is terminated when there is less than 5% change in the policy extracted from the action-value function and the policy performance analysis yields greater than 98% success.

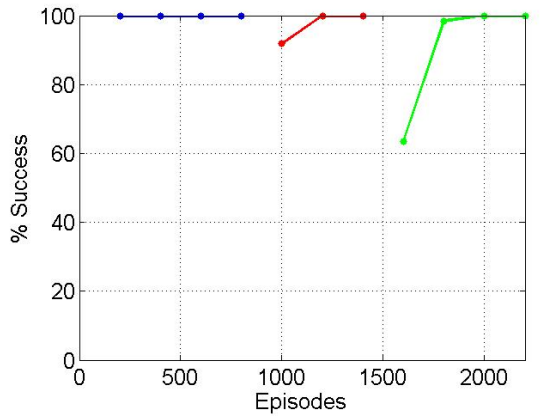

Fig. 60. Wing with AAG and PC: Monte Carlo Simulation Results

The figure shows that the first discretization reaches 100% within 200 episodes, and the learning is terminated after 800 episodes. The second and third levels of discretization both reach 100% success within 400 episodes, and the learning terminates for each level after 600 and 800 episodes, respectively. The agent only needed a total of 2200 episodes of the possible 15000 to fully converge on a usable policy. This is an 85% reduction in number of episodes.

2. Case 2: Q-Learning with GA and PC

The second case adds the genetic algorithm for function approximation to the Q-learning algorithm with PC and applies it to the morphing wing reinforcement learning problem. The problem for this example is set up similarly to that of Case 1. The learning is again paused and PC applied every 200 episodes. When learning is paused, the action-value function is approximated and PC applied to that approximation. Analysis of the learning is conducted in a similar manner as the previous case. Since only one level of discretization is learned, the final set of basis functions that level are presented, and the Monte Carlo simulation performance results are analyzed. The value function and policy are not presented.

a. Approximation

As described in the previous chapter, the GA determines the set of basis functions and degree or number of knots for those basis functions each time the learning is paused, in this case every 200 episodes. It is this approximation that is used for the policy comparison and performance analysis. In a sense the selections made by the GA evolves as the action-value function in tabular form evolves. As mentioned in previous chapters, each $Q(s, a_i)$ has a table of the learned action-values for that action, a_i. The GA is applied to each of these separately to yield $\tilde{Q}(s, a_i)$. Since there are eight actions for this problem, there are eight approximations that must be made to fully approximate the action-value function. Due to the computational limitation mentioned above, the gene representing the degree or number of nodes or centers is limited to a value of 6 or 0110. Listed in Table XXXVI are the bit strings encoding the approximations after the final episode. The basis functions for the end of the first and only level of discretization show consistency between many of

Table XXXVI. Approximation Bit Strings for Only Level of Discretization for Morphing Wing

	Level 1
$\tilde{Q}(s, a_1)$	1000110
$\tilde{Q}(s, a_2)$	1000100
$\tilde{Q}(s, a_3)$	1000110
$\tilde{Q}(s, a_4)$	1000110
$\tilde{Q}(s, a_5)$	1000100
$\tilde{Q}(s, a_6)$	1000110
$\tilde{Q}(s, a_7)$	1000110
$\tilde{Q}(s, a_8)$	1000110

the actions. $\tilde{Q}(s, a_2)$ and $\tilde{Q}(s, a_5)$ are both best approximated by linear RBFs with $(4+2)^4 = 6^4 = 1296$ evenly distributed centers. The rest of the $\tilde{Q}(s, a_i)$ are best approximated by linear RBFs with $8^4 = 4096$ evenly distributed knots or centers. These approximations are larger than those for the airfoil due to the larger number of state variables. This suggests that the concept of the GA for function approximation is significant, but some changes may be necessary when it is applied to larger problems in future research.

b. Monte Carlo Simulation

Figure 61 shows the results of the Monte Carlo simulation performance analysis for this problem using the approximated action-value function every 200 episodes. Learning is terminated when there is less than 5% change in the policy extracted from the approximated action-value function in Eq. 5.2 and the policy performance analysis

yields greater than 98% success.

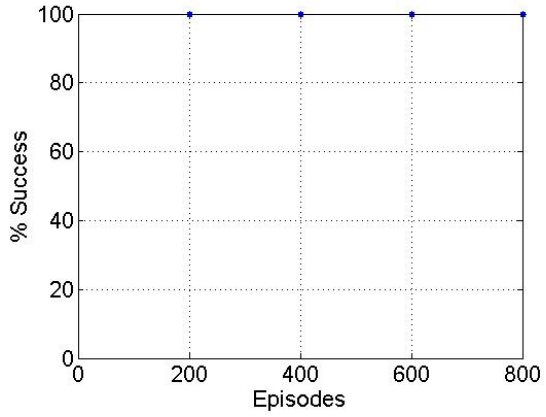

Fig. 61. Wing with GA and PC: Monte Carlo Simulation Results

The figure shows learning at this discretization reaches 100% within 200 episodes, and the learning is terminated after 800 episodes. Each approximated action-value function has 100% success. Based on the learned data here only 800 episodes are needed of the 5000 allowed episodes, which is an 84% reduction. Recall, that the GA is restricted for this problem do to computational concerns. The second gene can only go as high as 6 rather than the full 15 as originally designed. Despite this restriction the action-value function is approximated well enough to yield 100% success within 200 episodes. This suggests that allowing the GA to search for approximations with higher values encoded in the second gene may not be necessary for some problems. Restricting or modifying the GA in this manner may help keep the approximation itself a manageable size.

D. Summary

In this chapter the wing was cast as a reinforcement learning problem and run through various components of the algorithm developed in this research. Results show that the agent learned over a larger state-space than that presented in the previous chapter. The wing model has 13 possible shape changing parameters, and the agent learned using 4. Q-learning with multi-resolution state-space discretization and policy comparison was successfully demonstrated. The number of state-action pairs was reduced by two orders of magnitude from the state-space discretized at the finest level, and only 2200 episodes of the possible 15000 were needed to converge to a goal of $\left(\frac{C_L}{C_D}\right)_{\max} \pm 6$, which is an 85.3% reduction. Results also successfully demonstrate Q-learning with policy comparison and the genetic algorithm for function approximation. The action-value function was successfully approximated such that the extracted policy showed fully converged behavior within 800 episodes.

CHAPTER X

UNUSUAL APPLICATION EXAMPLE - THERMAL LOCATION FOR AUTONOMOUS SOARING

Large birds and glider pilots extend flight duration, increase cross-country speed, improve range, and conserve energy by thermals caused by convection in the lower atmosphere. The problem of learning an updraft field by an autonomous soaring UAV is cast as a reinforcement learning problem and tasked with learning the commands that produce the actions to move from some position to the middle of an updraft based on the vertical velocity of the surrounding air. The unique aspect of this chapter is the modification of AAG to allow the multi-resolution learning around *multiple* Regions Of Interest.

Section A discusses the thermal model to be cast as the environment of the reinforcement learning problem. Then the full thermal location reinforcement learning problem is described followed by a series of examples that exercise the many aspects of the problem. The numerical results for this application are fairly extensive. Results for learning an updraft field with a single thermal in 2-dimensions, multiple thermals in 2-dimensions, and multiple thermals in 3-dimensions are all presented.

A. Thermal Model

The application in question is an updraft model developed by NASA Dryden Flight Research Center for use in developing autonomous soaring UAVs.[185] Autonomous soaring is when a UAV uses updrafts to extend flight duration, increase speed, improve range, or conserve energy. The model was created using data collected at the National Oceanic and Atmospheric Administration Surface Radiation station. The model is a statistical representation of the convective velocity scale, w^*, and the convective

mixing-layer thickness, z_i, which were used to determine updraft size, vertical velocity profile, spacing, and maximum height.[185] The full development of the model can be found in Reference [185].

B. Thermal Location Cast as a Reinforcement Learning Problem

The updraft model is cast as a reinforcement learning problem by defining the model itself as the environment. The model uses Cartesian coordinate inputs, x and y (and z if desired), supplied by the reinforcement learning agent as its current state, to calculate the distance to each of the randomly spaced updraft centers, w_{c_i}, using simple Euclidean distance:

$$d_i = \sqrt{\left(w_{c_{i_x}} - x\right)^2 + \left(w_{c_{i_y}} - y\right)^2} \tag{10.1}$$

The minimum distance is extracted, and the vertical velocity at the current location is calculated and output to the agent. Figure 62 shows a representative updraft field. It is assumed the the agent is able to move throughout this updraft field and store learned information as to the location and path to updrafts.

The updraft model and the reinforcement learning agent interact significantly during both the learning stage, when the location and path to updrafts are learned, and the operational stage, when the agent is asked to move through the updraft field to the nearest updraft by transferring from state to state. The purpose of the reinforcement learning agent is to learn the series of actions necessary to command both the x- and y-positions of the agent to the nearest updraft. The two parts of the system interact as follows.

The agent interacts with this environment in a similar manner to the agents interacting with the airfoil and wing environments in the preceding chapters. For

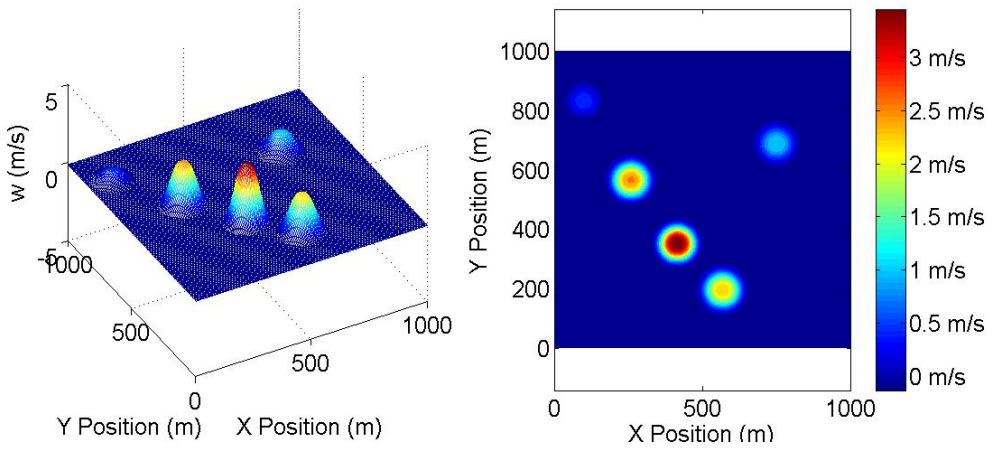

Fig. 62. Representative Updraft Field

Table XXXVII. Thermal Location Axis Definitions

x_i	Definition
x_1	X Position (m)
x_2	Y Position (m)
x_3	Z Position (m)

this problem, the state variables are the Cartesian coordinates and the actions are incremental changes in these coordinates. These state variables are listed in Table XXXVII.

It is noted that learning in this manner, moving from vertex to adjacent vertex, produces an action-value matrix specifying preferences for local transitions that could be used to create a path by specifying a series of actions from some state to a goal state. Specifying the actions in this way allows for the agent to learn to avoid hazards in the updraft field, such as downdrafts (negative vertical velocity of the air) or

ground obstacles (should they be included in the model). If actions were specified by transitions from one specific position or state to another, e.g. from $(x, y)_1 = (0m, 100m)$ to $(x, y)_2 = (800m, 750m)$ and thus hoping to encounter an updraft at point 2, the agent would then have a beginning and an end, but would need some form of path planning algorithm to chart a path from point 1 to point 2. The problem with doing so in this reinforcement learning problem is that the agent does not have *a priori* knowledge of where updrafts and hazards are located between the two points, which would make planning a path to avoid hazards or utilize updrafts difficult.

The *goal*, w_g, of the agent for this problem is defined by the vertical velocity of the thermals. The agent is essentially asked to find the center of any and all thermals in the updraft field. Position changes of the agent in the updraft field necessitates evaluation of the local vertical velocity of the surrounding air. Changes in vertical velocity and proximity to updrafts located during learning define the reward, which leads to the reward being based on two equations depending on where in the state-space the agent is located. Due to the nature of the updraft field, there is a uniform velocity, or sink velocity, outside of the actual updrafts as indicated by the dark blue field in Figure 62. Therefore a gradient based reward function dependent on vertical velocity is not feasible. In this region, the reward is instead based on proximity to the updrafts. The multi-resolution discretization method records the states encountered by the agent that are categorized as "goal" states. Once at least one goal state is found, the reward function in the sink region is

$$r_u = \frac{0.001}{g_f^m} \left(\sqrt{\sum_{i=1}^{n} \left(s_{g_i} - s_i\right)^2} - \sqrt{\sum_{i=1}^{n} \left(s_{g_i} - s_i'\right)^2} \right) \tag{10.2}$$

where r_u is the reward in the sink velocity region, s_g is the nearest goal state to the current state, s, s' is the next state according to the Q-learning algorithm, g_f^m is

Table XXXVIII. Initial Updraft Field Limits

Initial Limit	Lower	Upper
X Position	0	1000
Y Position	0	1000
Z Position	100	1000

the discretization factor to the power of the current level of fineness, m, and 0.001 is a scaling factor. Prior to finding that first goal state, rewards in this sink region are simply 0. As the learning progresses and more goal states are found, it is likely that the s_g used for a given state will change. The action-value function will update accordingly and will still converge in this area.

When the agent encounters and maneuvers within an updraft, the reward function is based on a vertical velocity gradient defined by the following equation:

$$r_w = |w_g - w_s| - |w_g - w'_s| \tag{10.3}$$

where r_w is the reward in the updraft, w_g is the goal vertical velocity, w_s is the vertical velocity in the current state, and w'_s is the vertical velocity in the next state determined by the Q-learning algorithm. The area of interest is the goal (of which there can be multiple) of $w = 4m/s$ and the associated initial range is $\pm 4m/s$. The initial boundary limits of the state-space are listed in Table XXXVIII.

1. Single Thermal

Updraft fields with only one thermal are tested first to perfect the application of AAG, PC, and MGAP to this problem. This is a simple problem and similar to the two state variable morphing airfoil from Chapter VIII. For this simple case, only the

two state variable problem is tested.

2. Multiple Thermals

Updraft fields with multiple thermals are then tested. Having multiple thermals in an updraft field means that there are multiple Regions Of Interest the agent must explore and learn. The method for learning in this manner is discussed in Chapter III, Section D. Using this method constitutes a more complex problem on which to demonstrate AAG, PC, and MGAP. As such both the two state variable and three state variable reinforcement learning problems are tested.

C. Numerical Results

The purpose of the numerical examples are to demonstrate the learning performance of the reinforcement learning agent utilizing the multi-resolution state-space discretization method, policy comparison and performance stopping criteria, and MGAP with single and multiple goal region(s). The updraft field to be learned is unique to each case. The location of the thermals(s) and their individual strengths are randomly generated by the updraft model. Not all of the updrafts will have the same maximum strength, and therefore may not have any states in the stricter goal ranges as the discretization becomes finer. This detail does not present a problem as the overall goal is to learn the location of and the path to any and all of the updraft centers and not just the strongest. The agent is allowed a possible 5000 episodes with which to explore the state-space of (x, y) or (x, y, z) combinations for each level of discretization in each Region Of Interest.

1. Single Thermal in 2-Dimensions Numerical Results

The simplest form of the thermal location for autonomous soaring reinforcement learning problem is single thermal in 2-dimensions, which means there are two state variables: X position and Y position. Three cases are shown in the following sections for this problem. First, the agent learns using the policy comparison and performance stopping criteria. The second case has the agent use AAG and PC to learn the updraft field. The final case has the agent use MGAP while learning. The three cases serve to show how the various components are applied to this problem as well as the benefit resulting from their application.

All three cases have the same admissible actions, discretization factor g_f, and number of level of discretization M. One additional factor is introduced for this problem that is not in the airfoil or wing reinforcement learning problems. This factor, g_w, is applied to the goal range, and subsequently the Region Of Interest, rather than g_f. After initial testing of this problem, this modification was added to aid learning. These parameters are listed in Table XXXIX. The goal, w_g, and initial range, g_{r_1}, are dependent on the maximum vertical velocity in the center of the thermal.

a. Case 1: Q-Learning with PC

PC is the first component added to Q-learning and applied to this problem. The updraft field to be learned is shown in Figure 63. The goal, as determined from the updraft field, is $w_g = 3m/s$, and the range is $g_r = 3m/s$. The learning performance is analyzed in much the same way as the learning on the airfoil and wing were analyzed, namely the Monte Carlo simulation results are considered as well as the final value function and extracted greedy policy.

Table XXXIX. Thermal Learning Parameters for Thermals in 2-Dimensions

Parameter	Value
Episodes	5000
α	0.01
γ	0.7
g_f	0.2
g_w	0.5
M	3
$\left(h^1_{x_1}, h^1_{x_2}\right)$	(50m, 50m)

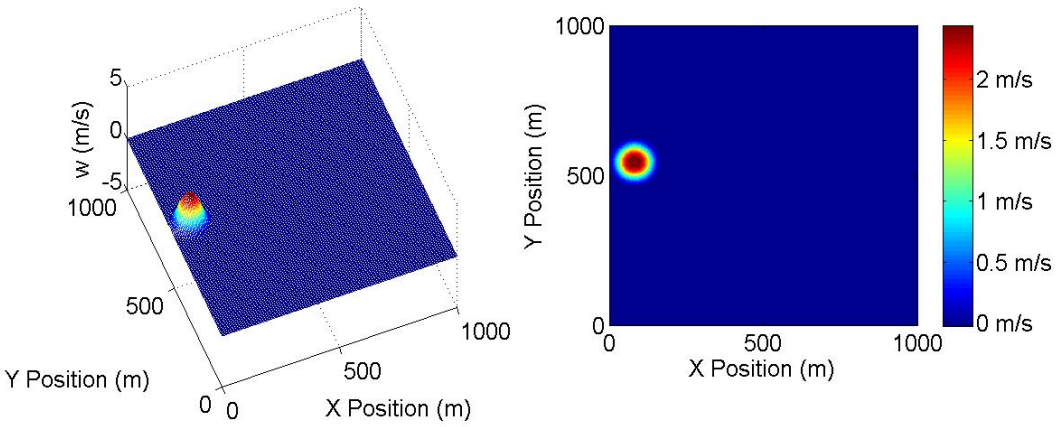

Fig. 63. Updraft Field for Single Thermal in 2-Dimensions Case 1

Monte Carlo Simulation: Figure 64 shows the results of the Monte Carlo simulation performance analysis for this problem. The one level of discretization is allowed a possible 5000 episodes. Recall that learning on each level of discretization is terminated when there is less than 5% change in the policy extracted from the action-value function and the policy performance analysis yields greater than 98% success.

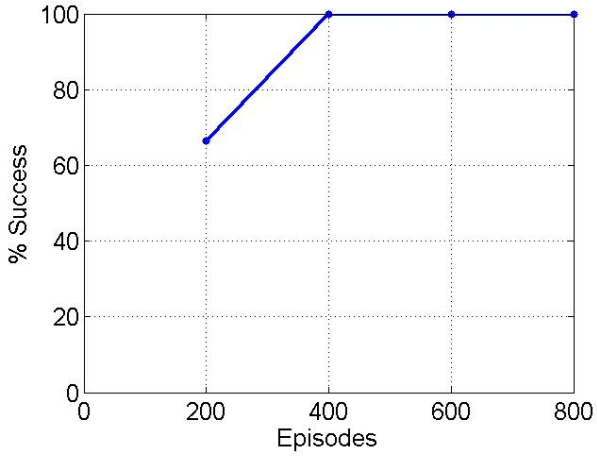

Fig. 64. Single Thermal in 2-Dimensions with PC: Monte Carlo Simulation Results

The figure shows that the policy reaches 100% success within 400 episodes, but does not terminate until 800 episodes have elapsed. This indicates that the policy still has at least 5% change every 200 episodes until the 800^{th} episode. However, 800 is still a small fraction from the user defined 5000 total episodes allowed the agent. The agent only needs 16% of the allowed episodes to find a converged and usable policy. This translates to an 84% reduction in number of episodes.

Value Function and Policy Analysis: Figure 65 shows the value function and greedy policy for this problem. There is a marked increase and crest in the value function that directly corresponds to the location of the thermal in the updraft field. Within

Table XL. Thermal Policy Color Scheme

Action	Color
$-h_{x_1}$	Blue
$+h_{x_1}$	Cyan
$-h_{x_2}$	Yellow
$+h_{x_2}$	Red

the circular crest in the value function there is a marked decrease in value toward the center of the thermal. This phenomenon is similar to what was seen in the airfoil learning cases and is a product of the reward functions used. The policy representation is color coded by action just as for learning on the airfoil. The color coding is repeated in Table XL for reference in this chapter. The greedy policy shown in Figure 65 shows definitively that the agent will move directly to the center of the thermal almost exclusively. There are a few areas that the agent will take a slight detour before proceeding to the center, and one spot within the thermal itself that the agent will get "stuck" within the thermal but not quite at the center, as shown by the yellow patch at $(x, y) = (100, 500)$. It is likely that this patch would change if learning were allowed to continue, but within the scope of this research and the capability of PC, these are good results for this problem and this component.

b. Case 2: Q-Learning with AAG and PC

Now that PC has been applied and tested with success, AAG is added to the Q-learning with PC algorithm and tested on the single thermal reinforcement learning problem. The problem for this example is set up similarly to that of Case 1. The parameters listed in Table XXXIX are used. The updraft field to be learned in this case

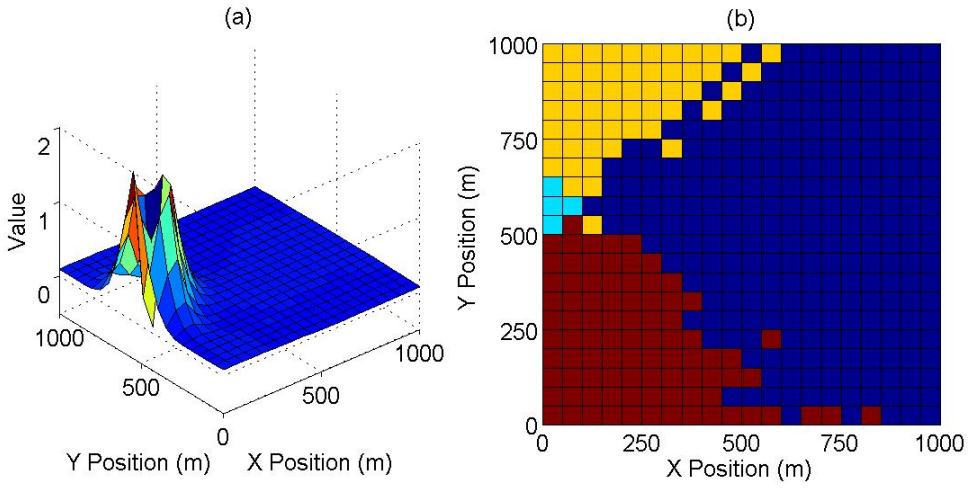

Fig. 65. Single Thermal in 2-Dimensions with PC: Value Function (a) and Policy Representation (b)

is shown in Figure 66. The goal, as determined from the updraft field, is $w_g = 4m/s$, and the initial range is $g_{r_1} = 4m/s$. The final range for the goal in this problem is $g_{r_3} = 1m/s$. The learning is paused and PC applied every 200 episodes. Analysis of the learning is conducted in the same manner as before with one addition: the dimensionality of the multi-resolution action-value function is compared with that of the full state-space discretized at the finest level, the Monte Carlo simulation performance results are analyzed, and the final value function and policy are considered.

Dimensionality: The dimensionality for this problem is analyzed in two ways. The first is visually considering the distribution of states in the state-space visited by the agent. The second is considering the number of states and state-action pairs as was done in previous chapters. Figure 67 shows the distribution of visited states for this problem. There are more states in and around the thermal as a result of the

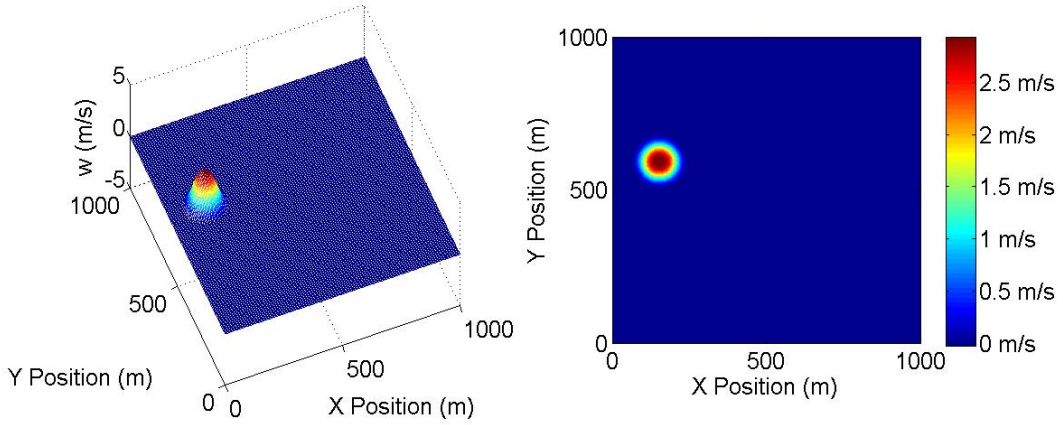

Fig. 66. Updraft Field for Single Thermal in 2-Dimensions Case 1

multi-resolution method with the densest packing of states around the center of the thermal. This change in state density shows how the Region Of Interest contracts for each subsequent finer discretization.

The numerical dimensionality is listed in Table XLI. The multi-resolution method reduces the number of state-action pairs the agent must visit by two orders of magnitude, which is a much larger reduction than what was seen in Chapter VIII with the airfoil. The reason is that the Region Of Interest in this problem is much smaller. AAG allows the agent to focus most of its attention on this one small Region Of Interest rather than wasting time in the rest of the state-space that contains nothing of interest, attempting to update over one million states to attain the same level of refinement.

Monte Carlo Simulation: Figure 68 shows the results of the Monte Carlo simulation performance analysis for this problem. Each level of discretization is allowed a possible 5000 episodes.

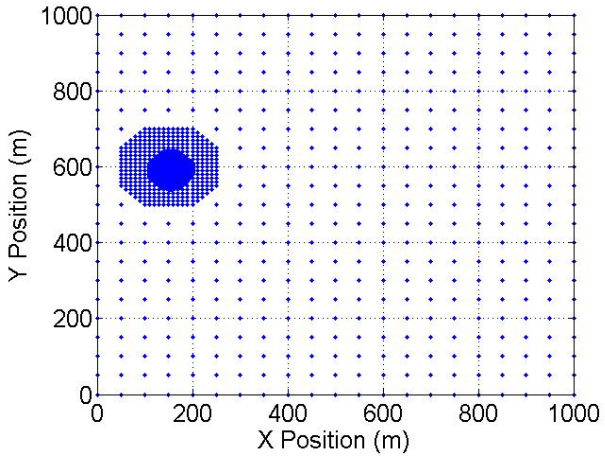

Fig. 67. Single Thermal in 2-Dimensions with AAG and PC: Visited States in the Updraft Field

Table XLI. States and State-Action Pairs

	States	State-Action Pairs
Multi-Resolution	2539	10156
Single-Resolution	251001	1004004

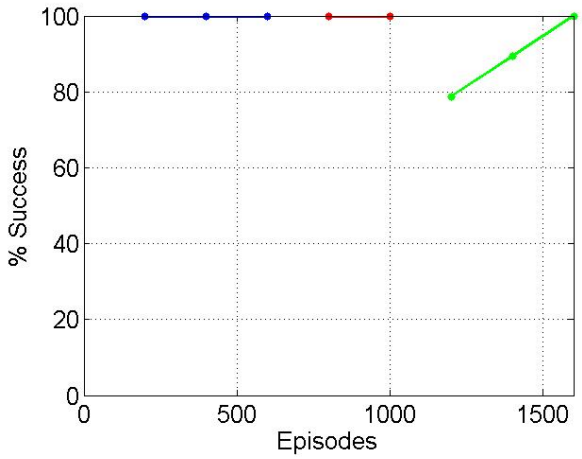

Fig. 68. Single Thermal in 2-Dimensions with AAG and PC: Monte Carlo Simulation Results

The figure shows that all three levels of discretization converge quickly. The first discretization reaches 100% within 200 episodes, and the learning is terminated after 600 episodes, 4400 fewer episodes than the total allowed 5000 episodes. The agent only needs 400 episodes to learn the second discretization to the point that there is $< 5\%$ change in the policy, and fewer than 200 episodes to reach 100% success. The third and final discretization reaches a performance measure of 100% success in 400 episodes as the learning is terminated. It is unusual that the learning is terminated just as the performance measure reaches 100%. In this case there a small number of states in the final discretization as compared to the total number of states in the entire state-space. It is possible that many of these are changing during each episode, but only account for 3% or 4% of the total number of states. That is why the performance analysis is included as a stopping criterion, for those cases when the policy has converged $> 95\%$ but has not yet converged to a usable policy. As a result only 1600 of the possible 15000 episodes were needed for the agent to learn a

policy with a small goal range of $g_{r_3} = 1m/s$, which is an 89% reduction in number of episodes.

Value Function and Policy Analysis: Figure 69 shows the value function and greedy policy for this problem. Again, there is a marked increase and crest in the value function that directly corresponds to the location of the thermal in the updraft field. The detail in the crest is more refined as a result of the finer discretizations in this region with the tell-tale blocky features further away from the crest resulting from the coarsest discretization. This same refinement is reflected in the policy representation. In the coarsely discretized region the policy shows the block nature of the policy, while near and in the thermal where the color coded actions meet the detail is more refined. Also, notice that there are no areas where the agent will get stuck as was evident in the previous case.

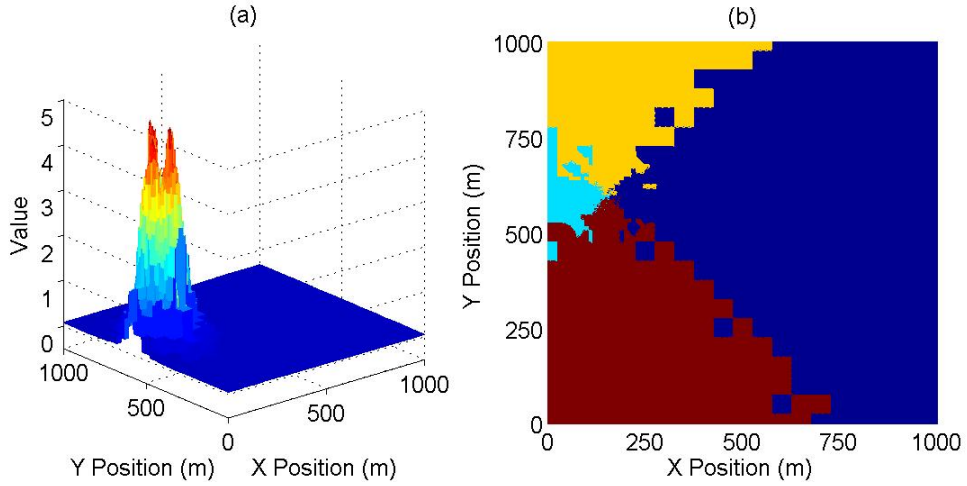

Fig. 69. Single Thermal in 2-Dimensions with AAG and PC: Value Function (a) and Policy Representation (b)

c. Case 3: Q-Learning with MGAP

The final case adds the genetic algorithm for function approximation to the Q-learning algorithm with AAG and PC and applies it to the two state variable reinforcement learning problem. The problem for this example is set up similarly to that of Case 2. The parameters listed in Table XXXIX are used. The updraft field to be learned in this case is shown in Figure 70. The goal, as determined from the updraft field, is $w_g = 3m/s$, and the initial range is $g_{r_1} = 3m/s$. The final range for the goal in this problem is $g_{r_3} = 0.75m/s$. The learning is paused and PC applied every 200 episodes using the GA developed in Chapter IV. Analysis of Q-learning with MGAP is conducted in a similar manner as before: the final set of basis functions for each level of discretization are presented, the Monte Carlo simulation performance results are analyzed, and the final approximated value function and associated policy are considered.

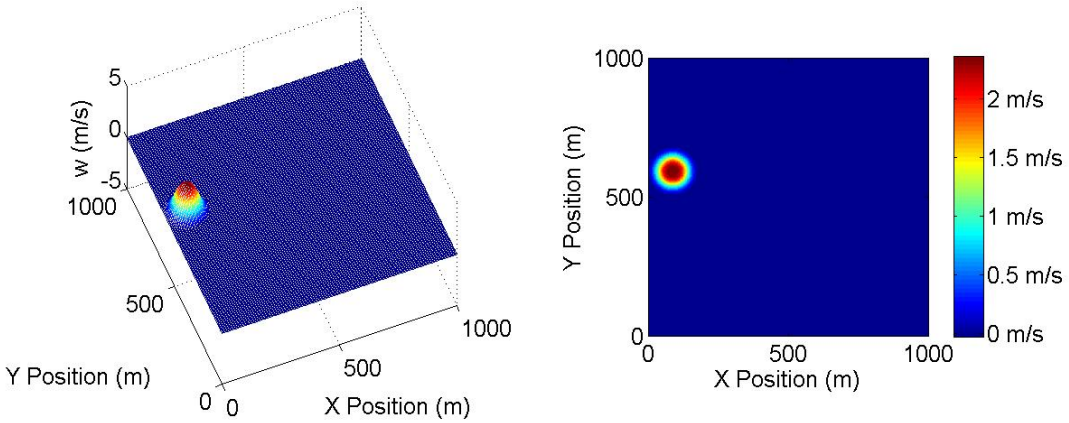

Fig. 70. Updraft Field for Single Thermal in 2-Dimensions Case 3

Figure 71 shows the distribution of visited states for this problem. There are more states in and around the thermal as a result of the multi-resolution method

with the densest packing of states around the center of the thermal. This change in state density shows how the Region Of Interest contracts for each subsequent finer discretization.

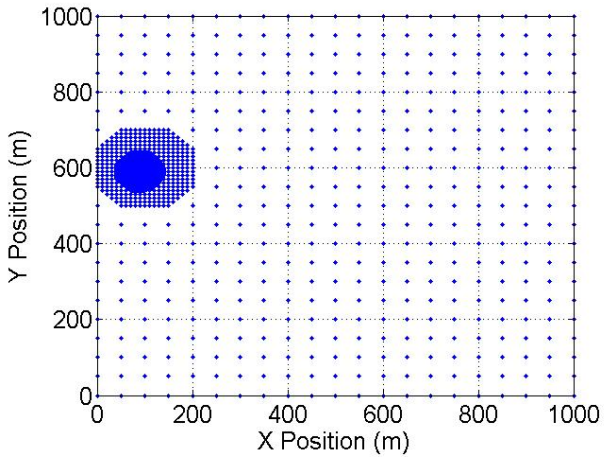

Fig. 71. Single Thermal in 2-Dimensions with MGAP: Visited State in the Updraft Field

Approximation: The GA determines the set of basis functions and degree or number of knots for those basis functions each time the learning is paused, in this case every 200 episodes. It is this approximation that is used for the policy comparison and performance analysis. As mentioned in previous chapters, each $Q(s, a_i)$ has a table of the learned action-values for that action, a_i. The GA is applied to each of these separately to yield $\tilde{Q}(s, a_i)$. Since there are four actions for this problem, there are four approximations that must be made to fully approximate the action-value function for this problem. Listed in Table XLII are the bit strings encoding the approximations after the final episode of each level of discretization.

The basis functions for the end of the first level of discretization show consistency between of each of the actions. $\tilde{Q}(s, a_1)$, $\tilde{Q}(s, a_2)$, and $\tilde{Q}(s, a_3)$ for this level are best

Table XLII. Approximation Bit Strings for Each Level of Discretization for Single Thermal in 2-Dimensions

	Level 1	Level 2	Level 3
$\tilde{Q}(s, a_1)$	1001110	1011110	1011011
$\tilde{Q}(s, a_2)$	1001011	1011111	1011110
$\tilde{Q}(s, a_3)$	1001110	1011101	1011110
$\tilde{Q}(s, a_4)$	1011110	1011110	1011110

approximated, according to the GA, by linear RBFs with 256, 169, and 256 evenly distributed centers, respectively. The approximation output by the GA for $\tilde{Q}(s, a_4)$ is cubic RBFs with 256 centers. The approximations for the end of the second level of discretization are more consistent. Each $\tilde{Q}(s, a_i)$ for this level is approximated by cubic RBFs, though each $\tilde{Q}(s, a_i)$ requires a different number of centers for the approximation, namely 256, 289, 225, and 256 centers for $i = 1, 2, 3, 4$, respectively. The approximation for the final action-value function is very similar to the second. The GA states that all four $\tilde{Q}(s, a_i)$ are best approximated by cubic RBFs with $\tilde{Q}(s, a_1)$ requiring 169 centers and the rest requiring 256 centers. These numbers are summarized in Table XLIII.

As was done for the airfoil example, consider the final approximations as compared to the tabulated action-value function. There are 2783 states and thus 11132 state-action pairs with associated preferences or action-values in the discretized action-value function. If only the final approximation is needed, then only the locations of only 425 centers and 937 weights must be stored for the approximation. That is a total of 1362 numbers that must be stored and used, which is 87% less than that for the discretized action-value function. If the approximations from the end of learning

Table XLIII. Comparison of Data for Tabulated and Approximated Action-Value Function for Single Thermal in 2-Dimensions

	Level 1 Discretization		Level 2 Discretization		Level 3 Discretization		
	Centers	Weights	Centers	Weights	Centers	Weights	States
$Q(s, a_1)$	256	256	256	256	169	169	2783
$Q(s, a_2)$	169	169	289	289	256	256	2783
$Q(s, a_3)$	256	256	225	225	256	256	2783
$Q(s, a_4)$	256	256	256	256	256	256	2783
Totals	425	937	770	1026	425	937	11132

on all three discretizations are needed, then 939 center locations and 2900 weights are required. This is a maximum of 3839 numbers that must be stored and used when appropriate, which is still 65% less than that for the discretized action-value function.

Monte Carlo Simulation: Figure 72 shows the results of the Monte Carlo simulation performance analysis for this problem using the approximated action-value function every 200 episodes. Each level of discretization is again allowed a possible 5000 episodes.

The figure shows that all three levels of discretization converge quickly as for the case with AAG and PC only. The first discretization reaches 100% within 400 episodes, and the learning is terminated after 600 episodes. The agent only needs 600 episodes to learn the second discretization and fewer than 200 episodes to reach $> 98\%$ success. The third and final discretization reaches a performance measure of

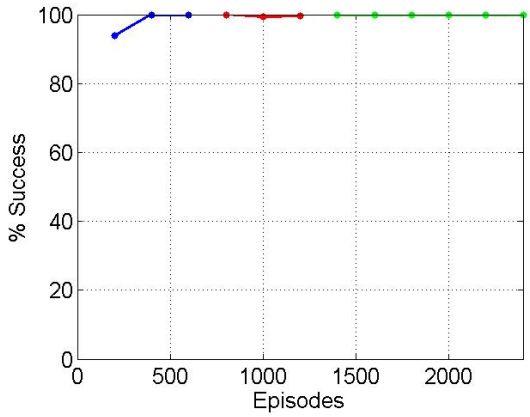

Fig. 72. Single Thermal in 2-Dimensions with MGAP: Monte Carlo Simulation Results

100% success in 200 episodes. The learning is not terminated until a full 1200 episodes ellapse at this level of discretization. This indicates that the approximate action-value is changing sufficiently to yield a $> 5\%$ change in policy every 200 episodes. It does settle, however, and learning is terminated with a total of only 2400 episodes needed, which is an 84% reduction with respect to the 15000 allowed episodes.

Value Function and Policy Analysis: The following figures show the value functions and greedy policies based on the approximated action-value functions for each level of discretization. These are determined using Eqs. 5.4 and 5.2. It is informative to consider each one individually to see how well the approximation captures or does not capture the global and local (i.e. in and around the Region Of Interest) behavior of the action-value function.

Figure 73 illustrates the value function and greedy policy based on the approximated action-value function after the final episode learned on the coarsest discretization, episode 600. The value function suggests that the action-value function is a fairly simple function to approximate. There is a smooth increase in value near the

thermal. The greedy policy shows this good approximation and supports the results from the Monte Carlo simulation. The color coded actions direct the agent to the center of the thermal within $w_g = 3 \pm 3m/s$ without hesitation.

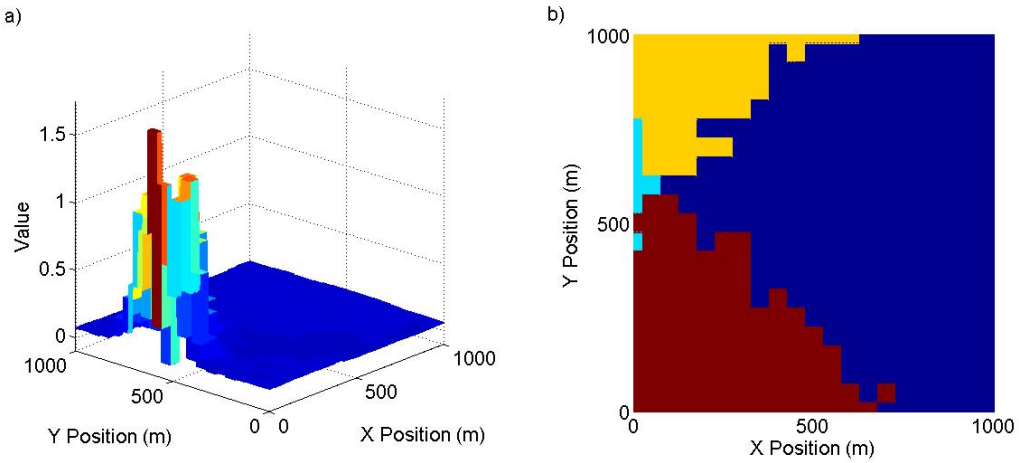

Fig. 73. Single Thermal in 2-Dimensions with MGAP, Episode 600: Approximate Value Function (a) and Policy Representation (b)

Figure 74 shows the value function and greedy policy based on the approximated action-value function after the final episode learned on the second discretization, episode 1200. The value function shows signs of the effects of the approximation. The peak around the thermal is consistent with that seen in Figure 73, but away from the location of the thermal there is evidence of oscillations in the value function. These oscillations are mirrored in the policy, which shows marked inconsistencies from the policy after episode 600 on the global scale. However, within the initial Region Of Interest that was rediscretized for the second level of learning, roughly defined by $0 \leq x_1 \leq 200$ and $500 \leq x_2 \leq 700$, the policy *is* consistent and shows refinement of the actions, evidence of the finer discretization.

Figure 75 shows the final value function and greedy policy based on the approx-

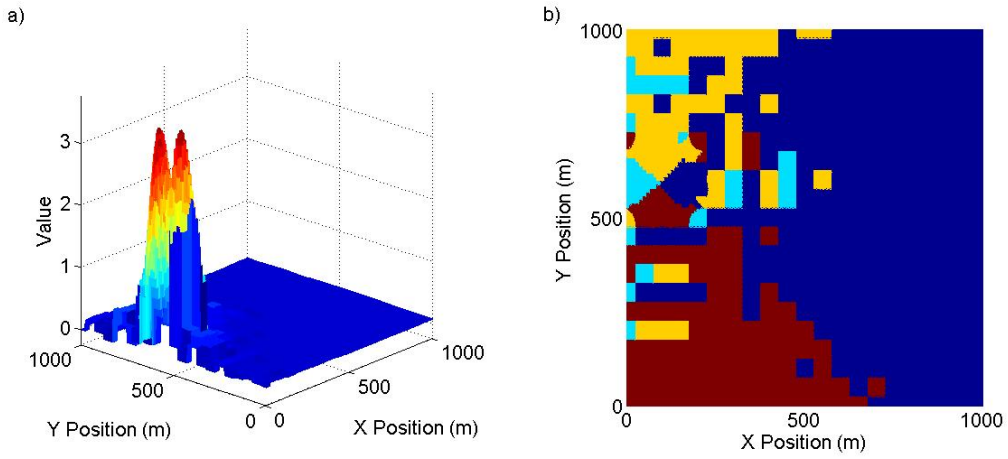

Fig. 74. Single Thermal in 2-Dimensions with MGAP, Episode 1200: Approximate Value Function (a) and Policy Representation (b)

imated action-value function for this problem. The region defined by $40 \leq x_1 \leq 140$ and $540 \leq x_2 \leq 640$ is discretized to the finest level. As was the case with the approximation after 1200 episodes, the area around the thermal is consistent with the previous two approximation examined, but not so away from the thermal. The oscillations in the outer regions are more pronounced. Again, the policy mirrors these oscillations. The final policy based on the approximated action-value function can only be used in and around the thermal itself, so in the region defined by $0 \leq x_1 \leq 200$ and $500 \leq x_2 \leq 700$. It does capture this small region in fine detail. Therefore, the agent can use the approximation after 600 episodes for global behavior away from the Region Of Interest and the final approximation in and around the Region Of Interest.

For comparison, Figure 76 shows the final value function and greedy policy determined by Eqs. 5.3 and 5.1 based on the tabular action-value function. The approximations after 600 episodes is consistent with the raw data in this figure. The

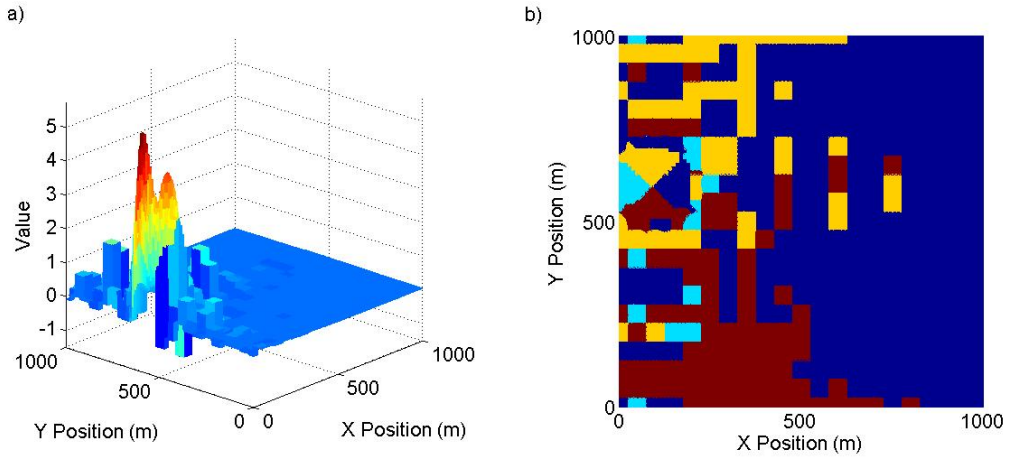

Fig. 75. Single Thermal in 2-Dimensions with MGAP, Episode 2400: Approximate Value Function (a) and Policy Representation (b)

approximation after 1200 episodes and the final approximation show good agreement with the region around the location of the thermal, but are not consistent with the raw data beyond that region.

2. Multiple Thermals in 2-Dimensions Numerical Results

The next step in the thermal location for autonomous soaring reinforcement learning problem is an updraft field with multiple thermals in 2-dimensions. Again, three cases are shown in the following sections for this problem. The agent learns using the policy comparison and performance stopping criteria. The second case has the agent use AAG and PC to learn the updraft field. The final case has the agent use MGAP while learning. The three cases serve to show the versatility of these components as applied to a state-space with multiple Regions Of Interest. The parameters for these cases are those listed above in Table XXXIX. The goal, w_g, and initial range, g_{r_1}, for this incarnation of the problem are dependent on the maximum vertical velocity in

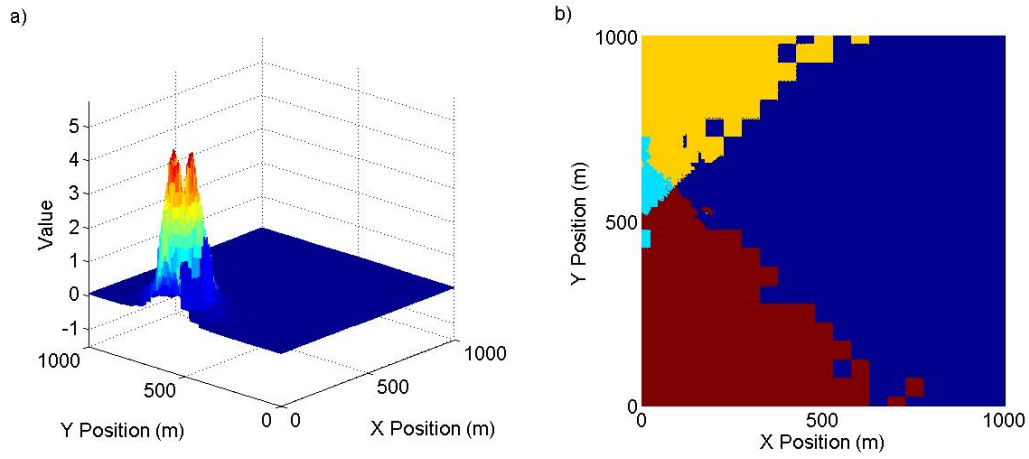

Fig. 76. Single Thermal in 2-Dimensions with MGAP, Episode 8600: Value Function (a) and Policy Representation (b)

the center of the thermal with the largest maximum vertical velocity.

a. Case 1: Q-Learning with PC

First consider the problem of learning multiple thermals with two state variables using Q-learning with PC. The updraft field to be learned is shown in Figure 77. The goal is determined from the strongest thermal in the updraft field and is $w_g = 4m/s$. Thus the range is $g_r = 4m/s$. The learning performance is analyzed from the Monte Carlo simulation results as well as the final value function and extracted greedy policy.

Monte Carlo Simulation: Figure 78 shows the results of the Monte Carlo simulation performance analysis for this problem. The one level of discretization is allowed a possible 5000 episodes. The figure shows that the policy reaches 100% success within 200 episodes, but does not terminate until 1200 episodes have elapsed. This indicates that the policy still has at least 5% change every 200 episodes, which is to be expected as there is likely activity and change around each thermal, as seen in the previous

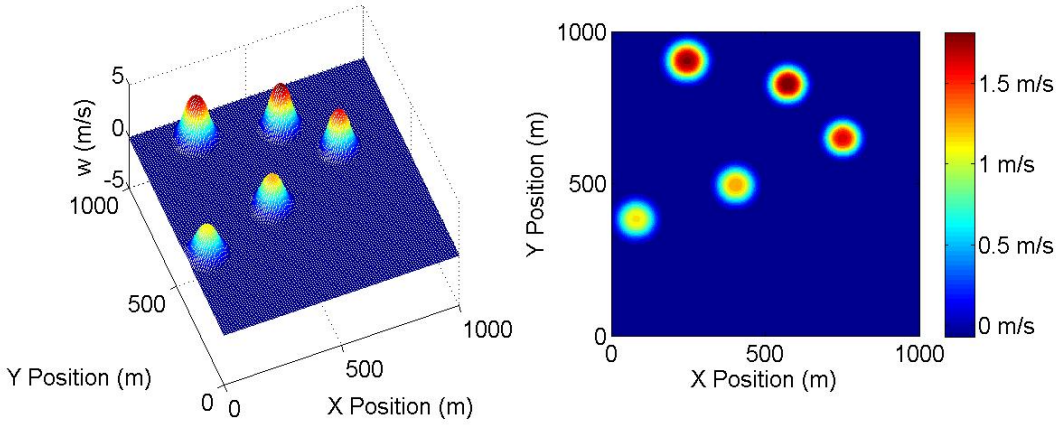

Fig. 77. Updraft Field for Multiple Thermals in 2-Dimensions Case 1

example. Nevertheless, the agent only needs 24% of the allowed episodes to find a converged and usable policy, which is a 74% reduction in number of episodes.

Value Function and Policy Analysis: Figure 79 shows the value function and greedy policy for this problem. There are crests around each thermal showing that the agent received positive reinforcement around each thermal. The relative magnitude of each crest reflects the relative strength of each thermal when compared to the updraft field in Figure 77. The two strongest thermals are located at $(x, y) = (250, 900)$ and $(x, y) = (575, 825)$. The corresponding crests in the value function in Figure 79 are of greater magnitude than the rest standing at about $V(s) = 4$.

The policy representation for this case is harder to read than the case with the single thermal. The various actions converge around each thermal. The learned policy tells the agent how to move to the nearest thermal from its current location. As for the single thermal case, there are a couple areas in the thermals that the policy will cause the agent to get "stuck" near the center and not allow it to move to the center

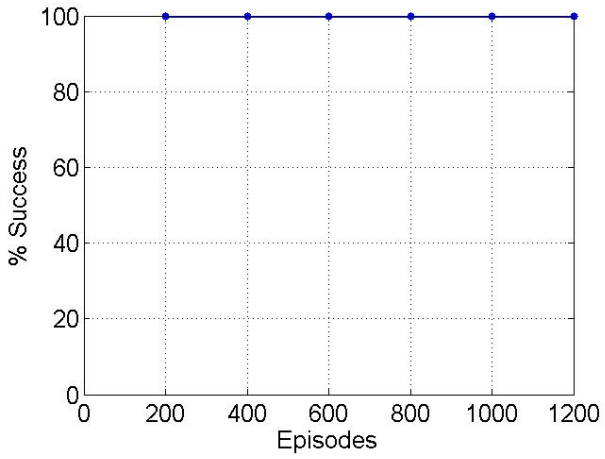

Fig. 78. Multiple Thermals in 2-Dimensions with PC: Monte Carlo Simulation Results

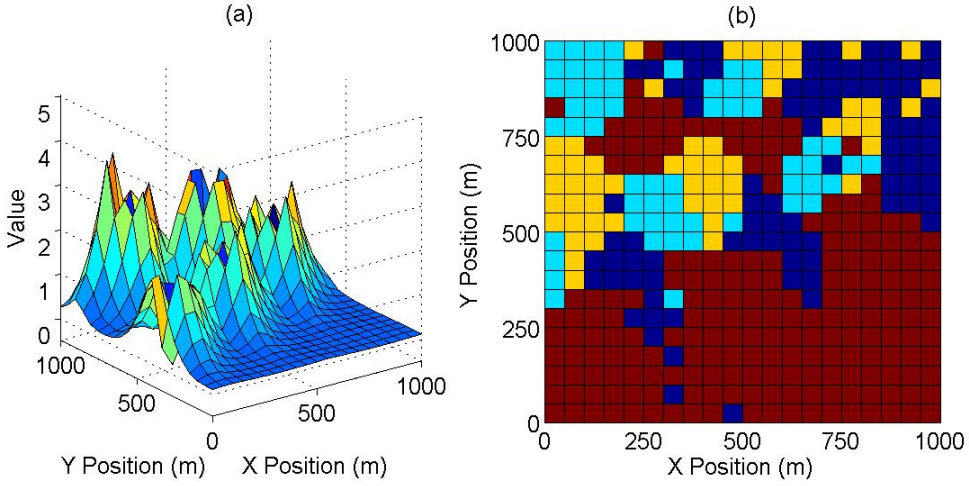

Fig. 79. Multiple Thermals in 2-Dimensions with PC: Value Function (a) and Policy Representation (b)

of the thermal. However, the agent did achieve the objective of learning the multiple thermal updraft field with one coarse level of discretization.

b. Case 2: Q-Learning with AAG and PC

Now that PC has been applied and tested with success for the multiple thermal case, AAG is added to the Q-learning with PC algorithm and tested. The parameters listed in Table XXXIX are used. The updraft field to be learned in this case is shown in Figure 80. The goal, as determined from the updraft field, is $w_g = 4m/s$, and the initial range is $g_{r_1} = 4m/s$. The final range for the goal in this problem is $g_{r_3} = 1m/s$. The learning is paused and PC applied every 200 episodes. Analysis of the learning is conducted in the same manner as for the second case with a single thermal: the dimensionality of the multi-resolution action-value function is compared with that of the full state-space discretized at the finest level, the Monte Carlo simulation performance results are analyzed, and the final value function and policy are considered.

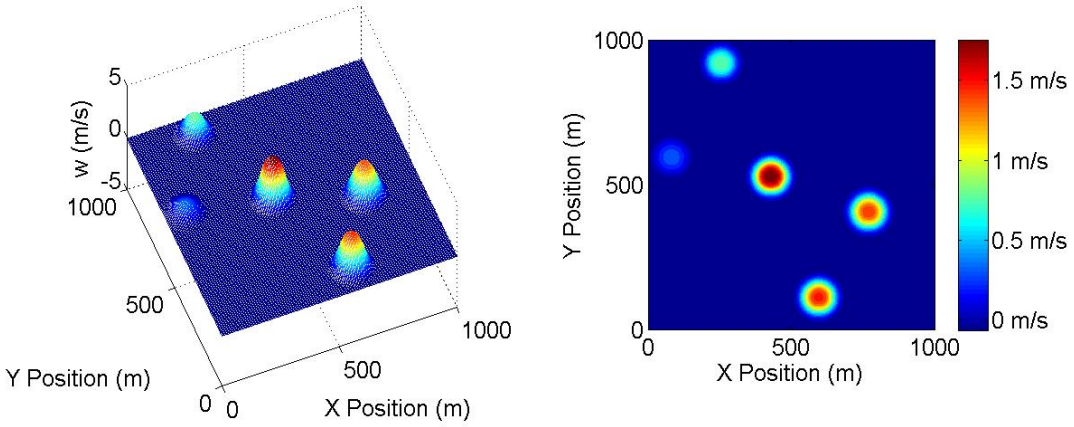

Fig. 80. Updraft Field for Multiple Thermals in 2-Dimensions Case 2

Dimensionality: The dimensionality for this problem is analyzed in the same two ways as before in the single thermal case. The first is visually considering the distribution of states in the state-space visited by the agent. The second is considering the number of states and state-action pairs. Figure 81 shows the distribution of visited states for this problem. There are more states in and around each thermal as a result of the multi-resolution method with the densest packing of states around the center of each thermal. Notice that the two thermals located at $(x, y) = (80, 590)$ and $(x, y) = (255, 920)$ do not have areas with states from the third level of discretization. The reason is that these two thermals are fairly weak (maximum velocities $< 1m/s$) compared to the strongest thermal, so at the higher levels of discretization with the more restricted goal regions, there are no goal states and thus no Region Of Interest to explore at the next level of discretization.

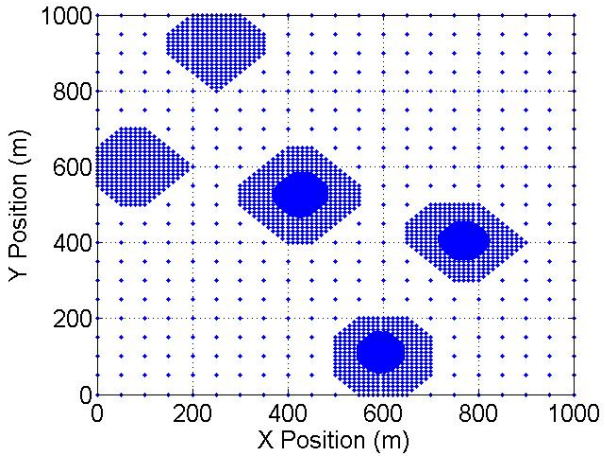

Fig. 81. Multiple Thermals in 2-Dimensions with AAG and PC: Visited State in the Updraft Field

The numerical dimensionality is listed in Table XLIV. The multi-resolution method again reduces the number of state-action pairs the agent must visit by two

Table XLIV. Multiple Thermals in 2-Dimensions with AAG and PC: States and State-Action Pairs

	States	State-Action Pairs
Multi-Resolution	7845	31380
Single-Resolution	251001	1004004

orders of magnitude. Even though there are now five Regions Of Interest, the combined area is still smaller than that seen with the airfoil. AAG allows the agent to focus most of its attention on these smaller Regions Of Interest individually rather than wasting time in areas of the state-space between these regions.

Monte Carlo Simulation: Figure 82 shows the results of the Monte Carlo simulation performance analysis for this problem. Each level of discretization is allowed a possible 5000 episodes, which means that the agent is allowed 5000 episodes for each Region Of Interest at each level of discretization. The thermal locations and their corresponding designation in Figure 82 are listed in Table XLV. Each Region Of Interest is tested separately for the second and third discretization. PC registers a success for the initial discretization if the agent moves from its initial state to *any* Region Of Interest.

The figures show that learning on the first level of discretization converges quickly, within 400 episodes. The first level is not terminated, however, until after 1400 episodes. The reasoning for this delay is the same as before. The policy is still changing by at least 5% every 200 episodes. It is not until the 1400^{th} episode that the policy has converged and has a performance of 100% success. The policy converges even more quickly for the second level of discretization for each thermal. The policy for each reaches 100% success almost immediately, and learning termi-

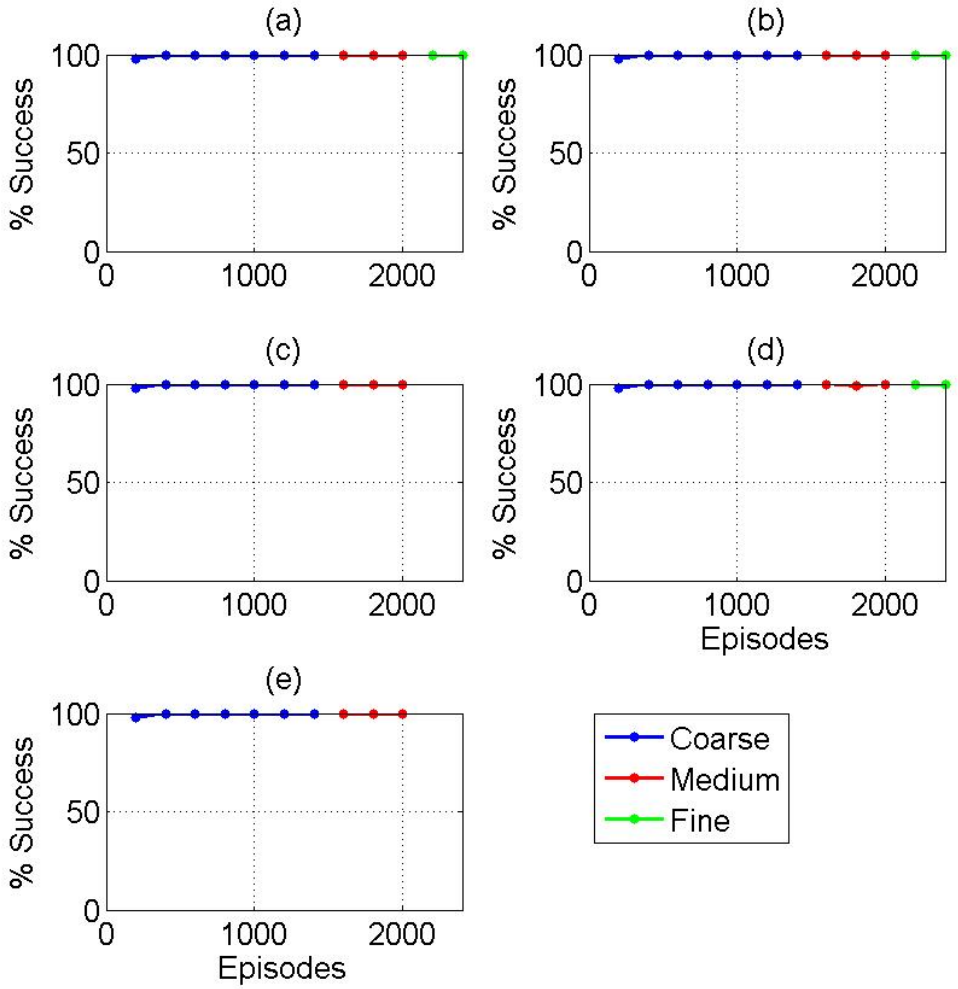

Fig. 82. Multiple Thermals in 2-Dimensions with AAG and PC: Monte Carlo Simulation Results

Table XLV. Multiple Thermals in 2-Dimensions with AAG and PC: Thermal Locations

Designation	X Position (m)	Y Position (m)
(a)	430	530
(b)	770	405
(c)	80	595
(d)	595	110
(e)	255	920

nates in 600 episodes. Only three of the thermals, (a), (b), and (d), needed further discretization and learning based on the final goal range of $g_{r_3} = 1m/s$. The figures show that the policy again converges almost immediately, in 200 episodes, and learning is terminated after 400 episodes. Table XLVI summarizes how the total possible number of episodes is determined. According to the table, only 5600 were needed of the total 90000 episodes possible, which is a 93.7% reduction.

Value Function and Policy Analysis: In both images in Figure 83, the various levels of discretization are evident in the blocky nature of the surfaces. The large squares in the policy representation and the large block structures in the value function representation indicate areas that the largest discretization was applied. The areas with finer discretization, especially around the bottom left region, are evident in the more refined surface gradation and color separation in the value function and policy, respectively. As in the previous case, each crest in the value function mirrors the strength of the thermal in that location. With the additional episodes at the finer levels of discretization, there are no obvious areas that would cause the agent to get stuck and

Table XLVI. Multiple Thermals in 2-Dimensions with AAG and PC: Number of Episodes Used

Discretization	Regions Explored	Episodes per Region	Total Episodes per Level	Total Episodes Allowed per Level
Level 1	1	1400	1400	5000
Level 2	5	600	3000	25000
Level 3	3	400	1200	15000
Total Episodes	–	–	5600	90000

be unable to proceed to the nearest thermal.

c. Case 3: Q-Learning with MGAP

The final step for this problem as for the last is to add the genetic algorithm for function approximation to the Q-learning algorithm with AAG and PC and demonstrate it with the two state variable multiple thermal reinforcement learning problem. The problem for this example is set up similarly to that of Case 2. The parameters listed in Table XXXIX are used. The updraft field to be learned in this case is shown in Figure 84. The goal, as determined from the updraft field, is $w_g = 5m/s$, and the initial range is $g_{r_1} = 5m/s$. The final range for the goal in this problem is $g_{r_3} = 1.25m/s$. The learning is paused and PC applied every 200 episodes using the GA developed in Chapter IV. Analysis of Q-learning with MGAP is conducted in a similar manner as before: the final set of basis functions for each level of discretization are presented, the Monte Carlo simulation performance results are analyzed, and the final approximated value function and associated policy are considered.

Figure 85 shows the distribution of visited states for this problem. There are

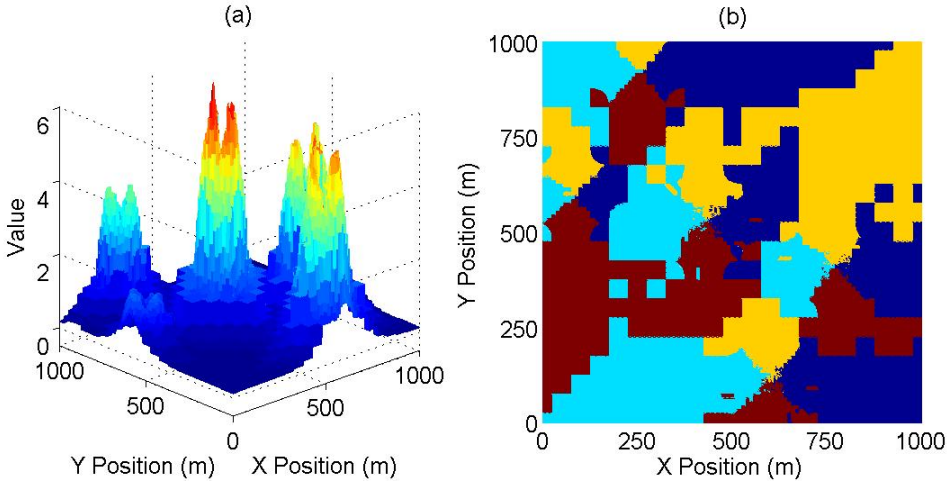

Fig. 83. Multiple Thermals in 2-Dimensions with AAG and PC: Value Function (a) and Policy Representation (b)

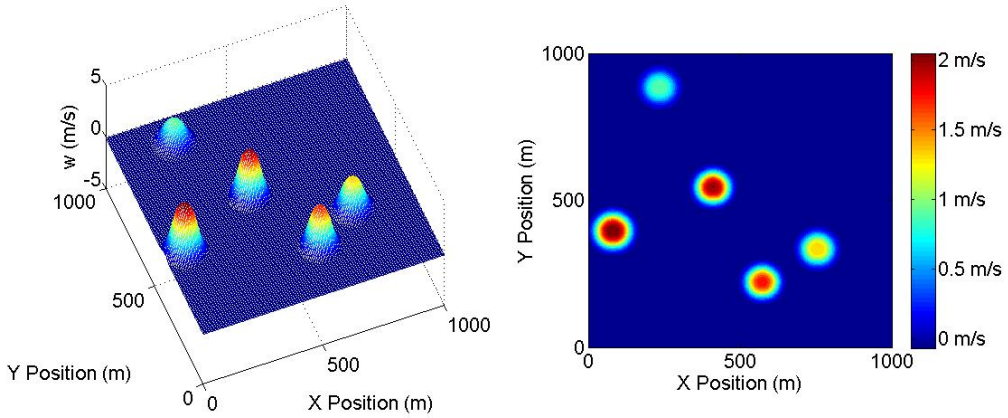

Fig. 84. Updraft Field for Multiple Thermals in 2-Dimensions Case 3

more states in and around the thermal as a result of the multi-resolution method with the densest packing of states around the center of the thermal. This change in state density shows how the Region Of Interest contracts for each subsequent finer discretization.

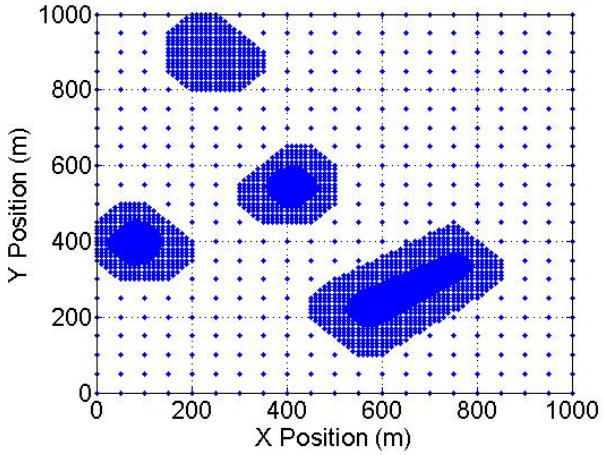

Fig. 85. Multiple Thermals in 2-Dimensions with MGAP: Visited State in the Updraft Field

Approximation: The GA determines the set of basis functions and degree or number of knots for those basis functions each time the learning is paused, in this case every 200 episodes. It is this approximation that is used for the policy comparison and performance analysis. As mentioned in previous chapters, each $Q(s, a_i)$ has a table of the learned action-values for that action, a_i. The GA is applied to each of these separately to yield $\tilde{Q}(s, a_i)$. Since there are four actions for this problem, there are four approximations that must be made to fully approximate the action-value function for this problem. Listed in Table XLVII are the bit strings encoding the approximations after the final episode of each level of discretization.

The basis functions for the end of the first level of discretization show consistency

Table XLVII. Approximation Bit Strings for Each Level of Discretization for Multiple Thermals in 2-Dimensions

	Level 1	Level 2	Level 3
$\tilde{Q}(s, a_1)$	1011110	1011110	1011110
$\tilde{Q}(s, a_2)$	1011110	1011111	1011110
$\tilde{Q}(s, a_3)$	1011110	1011110	1011110
$\tilde{Q}(s, a_4)$	1001110	1011111	1011110

between of each of the actions. $\tilde{Q}(s, a_1)$, $\tilde{Q}(s, a_2)$, and $\tilde{Q}(s, a_3)$ for this level are best approximated, according to the GA, by cubic RBFs with 256 evenly distributed centers. The approximation output by the GA for $\tilde{Q}(s, a_4)$ is linar RBFs with 256 centers. The approximations for the end of the second level of discretization are even more consistent. Each $\tilde{Q}(s, a_i)$ for this level is approximated by cubic RBFs with $\tilde{Q}(s, a_1)$ and $\tilde{Q}(s, a_3)$ requiring 256 centers and $\tilde{Q}(s, a_2)$ and $\tilde{Q}(s, a_4)$ requiring 289 centers. The approximation for the final action-value function is very similar to the second. The GA states that all four $\tilde{Q}(s, a_i)$ are best approximated by cubic RBFs with all of the $\tilde{Q}(s, a_i)$ requiring 256 centers. These numbers are summarized in Table XLVIII.

As was done for the single thermal example, consider the final approximations as compared to the tabulated action-value function. There are 10796 states and thus 43184 state-action pairs with associated preferences or action-values in the discretized action-value function. If only the final approximation is needed, then the locations of only 256 centers and 1024 weights must be stored for the approximation. That is a total of 1280 numbers that must be stored and used, which is 97% less than that for the discretized action-value function. If the approximations from the end of learning

Table XLVIII. Comparison of Data for Tabulated and Approximated Action-Value Function for Multiple Thermals in 2-Dimensions

	Level 1 Discretization		Level 2 Discretization		Level 3 Discretization		
	Centers	Weights	Centers	Weights	Centers	Weights	States
$Q(s, a_1)$	256	256	256	256	256	256	10796
$Q(s, a_2)$	256	256	289	289	256	256	10796
$Q(s, a_3)$	256	256	256	256	256	256	10796
$Q(s, a_4)$	256	256	289	289	256	256	10796
Totals	256	1024	545	1090	256	1024	43184

on all three discretizations are needed, then 545 center locations and 3138 weights are required. This is a maximum of 3683 numbers that must be stored and used when appropriate, which is still 91% less than that for the discretized action-value function.

Monte Carlo Simulation: Figure 86 shows the results of the Monte Carlo simulation performance analysis for this problem. Each level of discretization is allowed a possible 5000 episodes, which means that agent is allowed 5000 episodes for each Region Of Interest at each level of discretization. The thermal locations and their corresponding designation in Figure 86 are listed in Table XLIX. Each Region Of Interest is tested separately for the second and third discretization. PC registers a success for the initial discretization if the agent moves from its initial state to *any* Region Of Interest.

The figures show that learning on the first level of discretization converges quickly, within 600 episodes. The first level is not terminated, however, until after

Table XLIX. Multiple Thermals in 2-Dimensions with MGAP: Thermal Locations

Designation	X Position (m)	Y Position (m)
(a)	80	395
(b)	410	545
(c)	575	220
	755	335
(d)	235	885

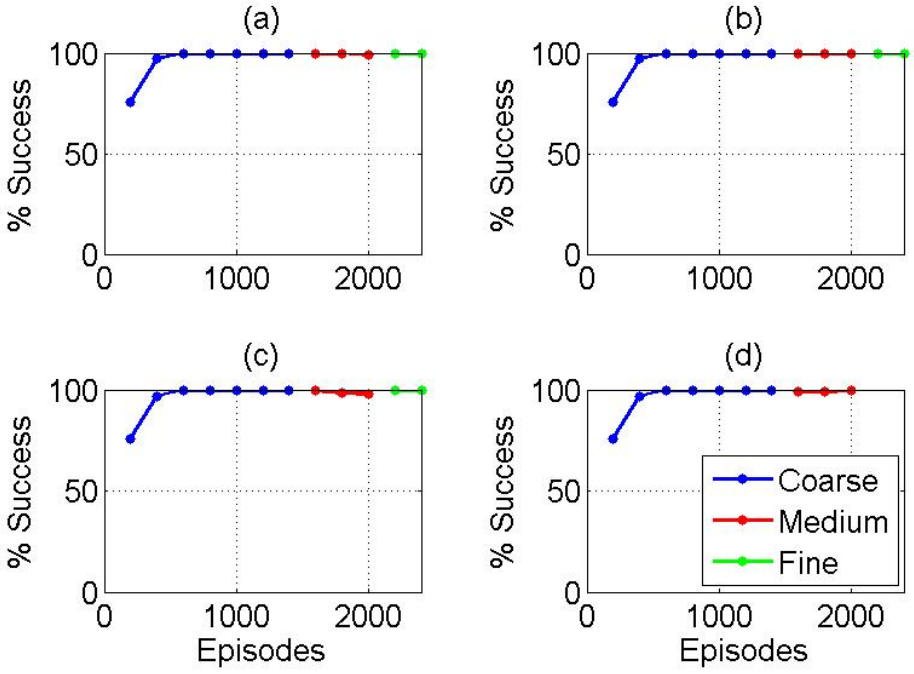

Fig. 86. Multiple Thermals in 2-Dimensions with MGAP: Monte Carlo Simulation Results

Table L. Multiple Thermals in 2-Dimensions with MGAP: Number of Episodes Used

Discretization	Regions Explored	Episodes per Region	Total Episodes per Level	Total Episodes Allowed per Level
Level 1	1	1400	1400	5000
Level 2	4	600	2400	20000
Level 3	3	400	1200	15000
Total Episodes	–	–	5000	85000

1400 episodes, as was the case for the algorithm with AAG and PC only. The reasoning for this delayis the policy is still changing by at least 5% every 200 episodes. It is not until the 1400^{th} episode that the policy has converged and has a performance of 100% success. The policy converges even more quickly for the second level of discretization for each thermal. The policy for each reaches $> 98\%$ success almost immediately, and learning terminates in 600 episodes. Only three of the thermals, (a), (b), and (c), needed further discretization and learning based on the final goal range of $g_{r_3} = 1.25m/s$. The figures show that the policy again converges almost immediately, in 200 episodes, and learning is terminated after 400 episodes. Table L summarizes how the total possible number of episodes is determined. According to the table, only 5000 were needed of the total 85000 episodes possible, which is a 94% reduction.

Value Function and Policy Analysis: The following figures show the value functions and greedy policies based on the approximated action-value functions for each level of discretization. These are determined using Eqs. 5.4 and 5.2. It is informative to consider each one individually to see how well the approximation captures or does

not capture the global and local (i.e. in and around the Region Of Interest) behavior of the action-value function.

Figure 87 illustrates the value function and greedy policy based on the approximated action-value function after the final episode learned on the coarsest discretization, episode 1400. The value function suggests that the action-value function is a fairly simple function to approximate despite the multiple thermals. There is a smooth increase in value around each thermal location. The greedy policy shows this good approximation and supports the results from the Monte Carlo simulation. The color coded actions direct the agent to the center of the nearest thermal within $w_g = 5 \pm 5m/s$ without hesitation.

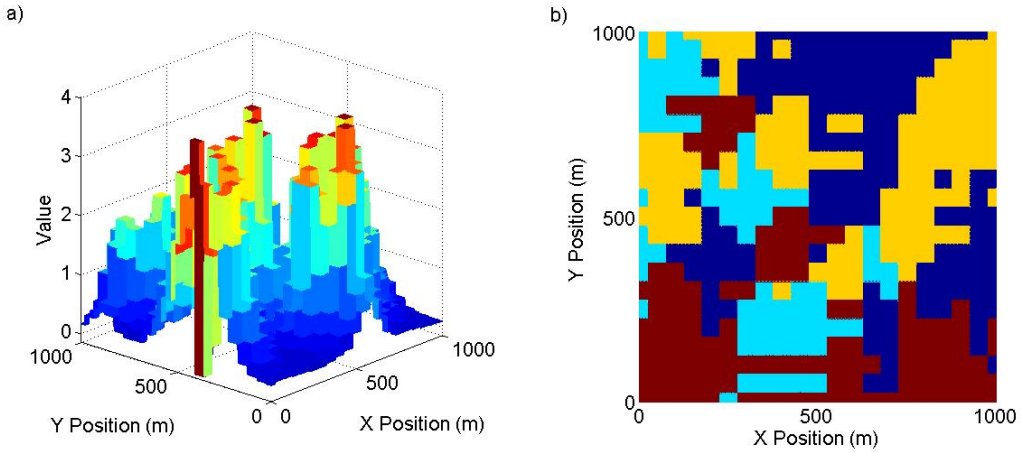

Fig. 87. Multiple Thermals in 2-Dimensions with MGAP, Episode 1400: Approximate Value Function (a) and Policy Representation (b)

Figure 88 shows the value function and greedy policy based on the approximated action-value function after the final episode learned on the second discretization, episode 2000. The value function shows evidence of the effects of the approximation. The peak around each thermal is consistent with that seen in Figure 87, but away from

the locations of the thermals and between each, there is evidence of oscillations in the value function. These oscillations are mirrored in the policy, which shows marked inconsistencies from the policy after episode 1400 on the global scale. However, within the initial Regions Of Interest around the locations of each thermal that were rediscretized for the second level of learning the policy *is* again consistent and shows refinement of the actions, evidence of the finer discretization.

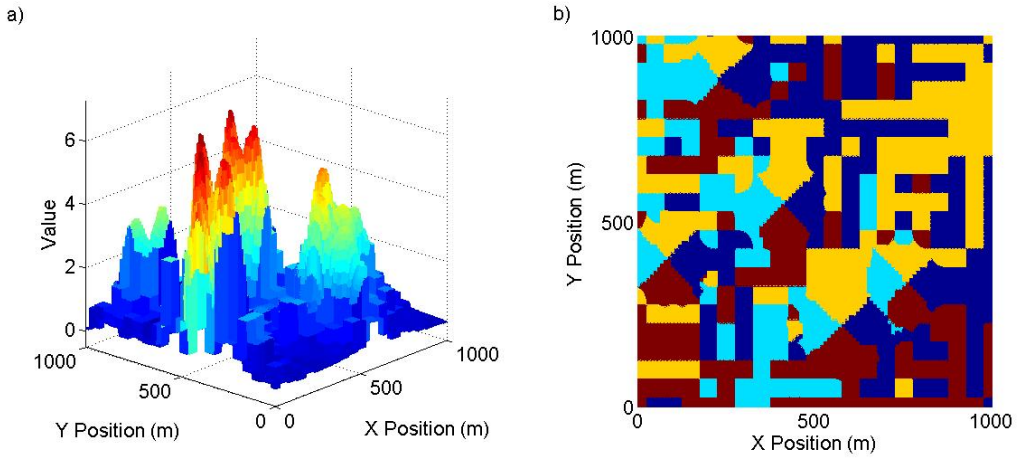

Fig. 88. Multiple Thermals in 2-Dimensions with MGAP, Episode 2000: Approximate Value Function (a) and Policy Representation (b)

Figure 89 shows the final value function and greedy policy based on the approximated action-value function for this problem. The interior regions of the thermals that require further learning are discretized to the finest level. As was the case with the approximation after 2000 episodes, the area around the center of each thermal is consistent with the previous two approximations examined, but not between the thermals. The oscillations in the these regions are more pronounced. Again, the policy mirrors these oscillations. The final policy based on the approximated action-value function can only be used in and around each thermal. It does capture each of these

small regions in fine detail. Therefore, the agent can use the approximation after 1400 episodes for global behavior away from the Regions Of Interest and the final approximation in and around the Regions Of Interest.

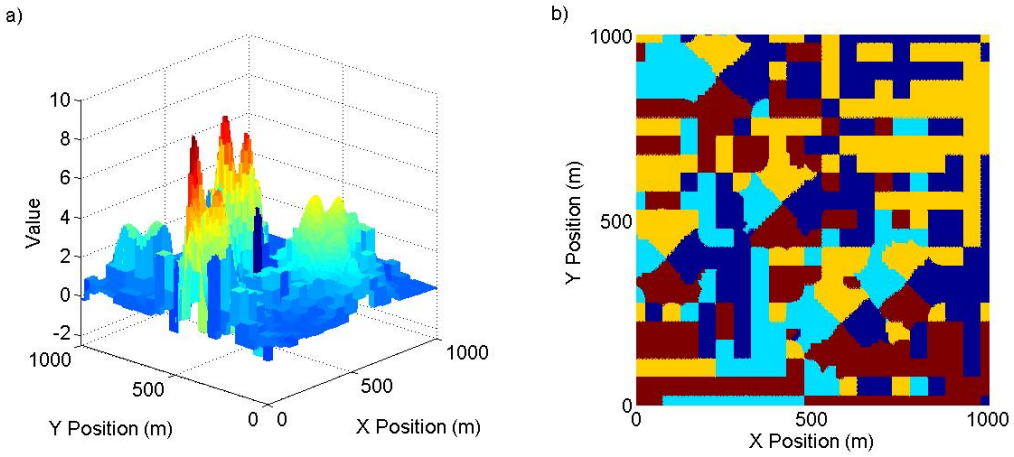

Fig. 89. Multiple Thermals in 2-Dimensions with MGAP, Episode 2400: Approximate Value Function (a) and Policy Representation (b)

For comparison, Figure 90 shows the final value function and greedy policy determined by Eqs. 5.3 and 5.1 based on the tabular action-value function. The approximations after 1400 episodes is consistent with the raw data in this figure. The approximation after 2000 episodes and the final approximation show good agreement with the regions around the locations of the thermals, but are not consistent with the raw data beyond those regions.

3. Multiple Thermals in 3-Dimensions Numerical Results

The final step in the thermal location for autonomous soaring reinforcement learning problem is an updraft field with multiple thermals in 3-dimensions. Only two cases are shown for this problem to avoid too much repitition. The first case has the agent

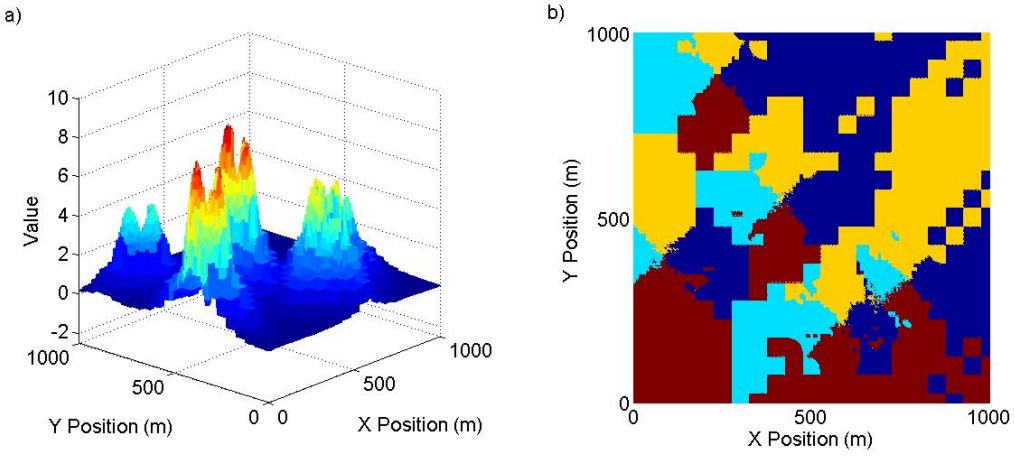

Fig. 90. Multiple Thermals in 2-Dimensions with MGAP, Episode 8600: Value Function (a) and Policy Representation (b)

use AAG and PC to learn the updraft field with 3 state variables. The second case has the agent use MGAP while learning. The three cases serve to show the versatility of these components as applied to a state-space with multiple Regions Of Interest. The parameters for these cases are listed in Table LI. As for the 2-dimensional case, the goal, w_g, and initial range, g_{r_1}, for this incarnation of the problem are dependent on the maximum vertical velocity in the center of the thermal with the largest maximum vertical velocity at $z = 300m$.

As for the morphing wing example, a case describing the results for the full MGAP algorithm applied to this problem is not included for one reason: the computational demands of the genetic algorithm for this larger problem were such that the supporting computer runs out of memory during the calculation of ω. Again, the case with the GA and PC is shown to illustrate show the success of these components.

Table LI. Thermal Learning Parameters for Thermals in 3-Dimensions

Parameter	Value
Episodes	5000
α	0.01
γ	0.7
g_f	0.5
g_w	0.5
M	3
$\left(h^1_{x_1}, h^1_{x_2}, h^1_{x_3}\right)$	$(100\text{m}, 100\text{m}, 100\text{m})$

a. Case 1: Q-Learning with AAG and PC

The first multiple thermal with three state variables case tests Q-learning with AAG and PC. The parameters listed in Table LI are used. The updraft field to be learned in this case is shown in Figure 91. This figure shows a cross-section of the full 3-dimensional updraft field at $z = 300m$. The goal, as determined from the updraft field, is $w_g = 5m/s$, and the initial range is $g_{r_1} = 5m/s$. The final range for the goal in this problem is $g_{r_3} = 1.25m/s$. The learning is paused and PC applied every 200 episodes. Analysis of the learning is conducted in a similar manner to the 2-dimensional field. As before, the dimensionality of the multi-resolution action-value function is compared with that of the full state-space discretized at the finest level, and the Monte Carlo simulation performance results are analyzed. The value function and policy are 4-dimensional surfaces and are difficult to visualize, so those results are not included in this analysis.

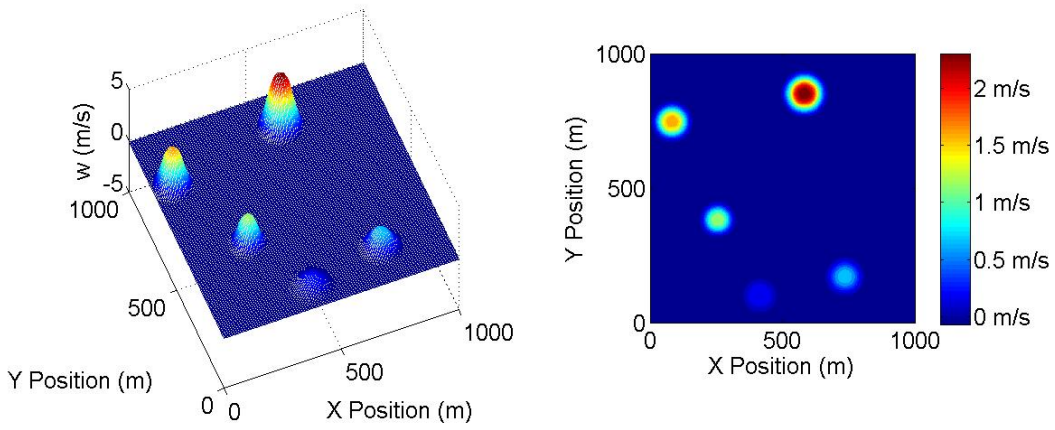

Fig. 91. Updraft Field for Multiple Thermals in 3-Dimensions Case 1

Dimensionality: The dimensionality for this problem is analyzed by visually considering the distribution of states in the state-space visited by the agent and considering the number of states and state-action pairs. Figure 92 shows the distribution of visited states for this problem. The figure shows the distribution in a 3-dimensional field and from a top down perspective. There are more states in and around each thermal as a result of the multi-resolution method with the densest packing of states around the center of each thermal. There is a noticeable flair in the distribution of states as z increases. This flair is a result of the bell shape of thermals created by the updraft model. Notice that the two thermals located at $(x, y) = (80, 745)$ and $(x, y) = (585, 850)$ are the only two with states from the third level of discretization. Also, the distribution of states around the two thermals located at $(x, y) = (100, 415)$ and $(x, y) = (735, 170)$ show no separation between the two. The two thermals are so close to each other that the method described in Chapter III, Section D could not distinguish between them and clustered them together.

The numerical dimensionality is listed in Table LII. The multi-resolution method

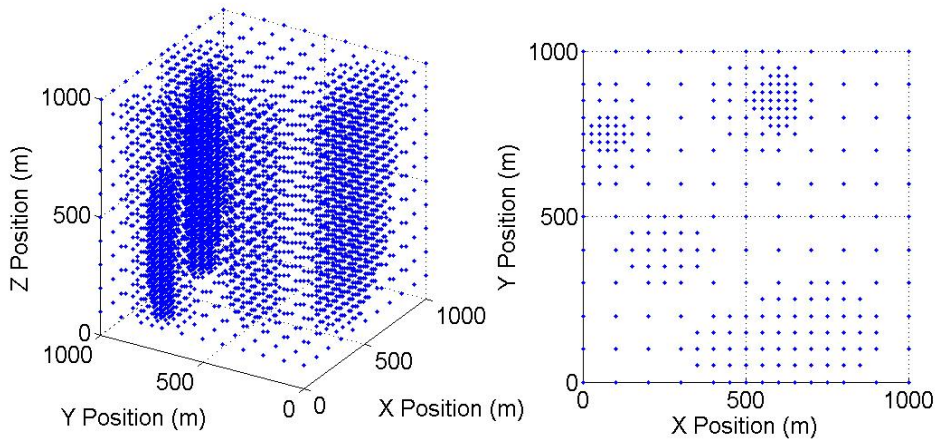

Fig. 92. Multiple Thermals in 3-Dimensions with AAG and PC: Visited State in the Updraft Field

Table LII. Multiple Thermals in 3-Dimensions with AAG and PC: States and State-Action Pairs

	States	State-Action Pairs
Multi-Resolution	4679	28074
Single-Resolution	62197	373182

reduces the number of state-action pairs the agent must visit by only one order of magnitude for this problem. This results from the larger initial discretization as well as the larger value for g_f. Regardless, AAG allows the agent to focus most of its attention on these four smaller Regions Of Interest individually rather than wasting time in areas of the state-space between these regions.

Monte Carlo Simulation: Figure 93 shows the results of the Monte Carlo simulation performance analysis for this problem. Each level of discretization is allowed a possible

Table LIII. Multiple Thermals in 3-Dimensions with AAG and PC: Thermal Locations

Designation	X Position (m)	Y Position (m)
(a)	255	380
(b)	100	415
	735	170
(c)	80	745
(d)	585	850

5000 episodes, which means that the agent is allowed 5000 episodes for each Region Of Interest at each level of discretization. The thermal locations and their corresponding designation in Figure 93 are listed in Table LIII. Note that (c) has two thermal centers listed. This means that when the goal states were clustered at the end of learning on the first level of discretization, the goal states around these two thermals not far enough away from each other to be recognized as separate thermals and are treated as one during learning on subsequent levels of discretization. Each Region Of Interest is tested separately for the second and third discretization. PC registers a success for the initial discretization if the agent moves from its initial state to *any* Region Of Interest.

The figures show that learning on the first level of discretization converges to 100% quickly, within 400 episodes. The first level is not terminated, however, until after 2200 episodes. The percent success varies between 98.4% and 100% for those final 1800 episodes of the first discretization. This variation is a consequence of the more complex field that must be learned. The policy converges to $> 98\%$ in 200 to 800 episodes for the second level of discretization. The thermals represented by (b) took the longest at 800 episodes to reach $> 98\%$ success. The reason is that with

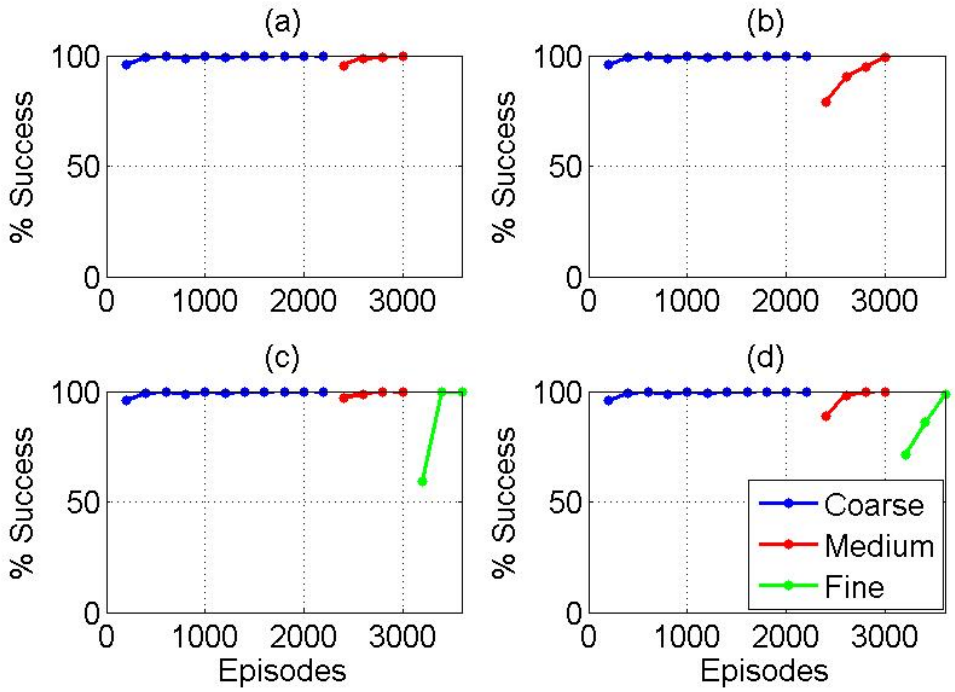

Fig. 93. Multiple Thermals in 3-Dimensions with AAG and PC: Monte Carlo Simulation Results

Table LIV. Multiple Thermals in 3-Dimensions with AAG and PC: Number of Episodes Used

Discretization	Regions Explored	Episodes per Region	Total Episodes per Level	Total Episodes Allowed per Level
Level 1	1	2200	2200	5000
Level 2	4	800	3200	20000
Level 3	2	600	1200	10000
Total Episodes	–	–	6600	80000

the two thermals in that Region Of Interest, there is a more complex updraft field to analyze than the other regions that contain only one thermal each. Only two of the thermals, (c) and (d), needed further discretization and learning based on the final goal range of $g_{r_3} = 1.25m/s$. The figures show that the policy converges in 400 to 600 episodes, and learning is terminated after 600 episodes. Table LIV summarizes how the total possible number of episodes is determined. According to the table, only 6600 were needed of the total 80000 episodes possible, which is a 91.8% reduction.

b. Case 2: Q-Learning with GA and PC

The final case adds the genetic algorithm for function approximation to the Q-learning algorithm with PC and applies it to the three state variable reinforcement learning problem. The problem for this example is set up similarly to that of Case 1. The parameters listed in Table XXXIX are used. The updraft field to be learned in this case is shown in Figure 94. The goal, as determined from the updraft field, is $w_g = 4m/s$, and the range is $g_{r_1} = 4m/s$. The learning is paused and PC applied every 200 episodes using the GA developed in Chapter IV. Analysis of Q-learning

with the GA and PC is conducted in a similar manner as before: the final set of basis functions are presented and the Monte Carlo simulation performance results are analyzed.

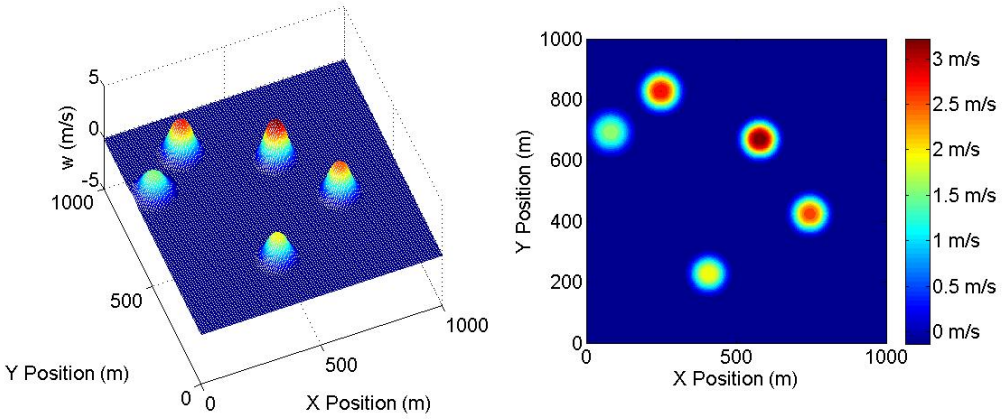

Fig. 94. Updraft Field for Multiple Thermals in 3-Dimensions Case 2

Approximation: As described in previous chapters, the GA determines the set of basis functions and degree or number of knots for those basis functions each time the learning is paused, in this case every 200 episodes. These approximations are used for the policy comparison and performance analysis. Since there are six actions for this problem, there are six approximations that must be made to fully approximate the action-value function. Due to the computational limitation mentioned earlier, the agent is restricted to learning on only one level of discretization. Listed in Table LV are the bit strings encoding the approximations after the final episode.

The basis functions for the end of the first and only level of discretization show gene combinations representing several basis function sets for the approximations. $\tilde{Q}(s, a_1)$ calls for linear RBFs with $(12+2)^3 = 14^3 = 2744$ evenly distributed centers.

Table LV. Approximation Bit Strings for Only Level of Discretization for Multiple Thermals in 3-Dimensions

	Level 1
$\tilde{Q}(s, a_1)$	1001100
$\tilde{Q}(s, a_2)$	0101110
$\tilde{Q}(s, a_3)$	0101110
$\tilde{Q}(s, a_4)$	1001101
$\tilde{Q}(s, a_5)$	0101110
$\tilde{Q}(s, a_6)$	0101110

$\tilde{Q}(s, a_2)$, $\tilde{Q}(s, a_3)$, $\tilde{Q}(s, a_5)$, and $\tilde{Q}(s, a_6)$ are all best approximated, according to the GA, by simple linear interpolation between $16^3 = 4096$ evenly distributed nodes. $\tilde{Q}(s, a_4)$ calls for linear RBFs with $15^3 = 3375$ evenly distributed centers. These approximations are larger than those for the two state variable problem in this chapter due to the extra of state variable.

Monte Carlo Simulation: Figure 95 shows the results of the Monte Carlo simulation performance analysis for this problem using the approximated action-value function every 200 episodes. Learning is terminated when there is less than 5% change in the policy extracted from the approximated action-value function in Eq. 5.2 and the policy performance analysis yields greater than 98% success.

The figure shows that learning at this discretization converges to 100% quickly, within 600 episodes. Learning is not terminated, however, until after 1600 episodes. The percent success remains 100% for those final 1000 episodes. Based on the learned data here only 600 episodes are needed of the 5000 allowed, which is an 88% reduction.

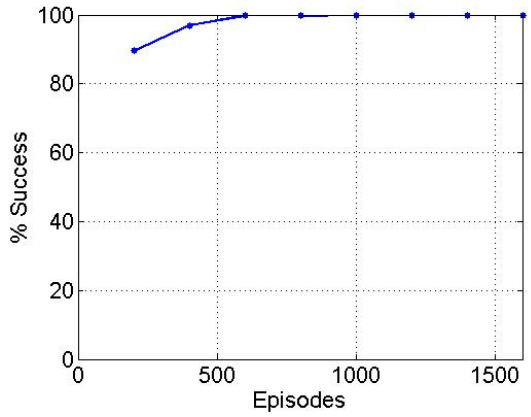

Fig. 95. Multiple Thermals in 3-Dimensions with GA and PC: Monte Carlo Simulation Results

D. Summary

In this chapter the thermal location problem with one or more Regions Of Interest was cast as a reinforcement learning problem and run through various components of the algorithm developed in this research with both two and three state variables. Results show that the agent autonomously determines where the multiple Regions Of Interest are located and continues learning within those regions. Q-learning with multi-resolution discretization and policy comparison was successfully demonstrated for the 2-dimensional single thermal case, the 2-dimensional multiple thermal case, and the 3-dimensional multiple thermal case. The agent was able to learn the goals of $w_g = 4 \pm 1m/s$, $w_g = 4 \pm 1m/s$, and $w_g = 5 \pm 1.25m/s$ for these three problems. Each showed significant reductions in the number of episodes required (89%, 93.7%, and 91.8%, respectively) and at least an order of magnitude reduction in the number of state-action pairs needed to learn these specific goals. It was also shown that the agetn autonomously decided which Regions Of Interest required additional learning and which regions held no additional information. Results also successfully

demonstrate Q-learning with MGAP for the 2-dimensional single thermal case and 2-dimensional multiple thermal case. The final Regions Of Interest for these two problems are $w_g = 3 \pm 0.75m/s$ and $w_g = 5 \pm 1.25m/s$, respectively. The actiona-value functions were successfully approximated and the extracted greedy policy converged to a usable policy within 2400 episodes and 5000 episodes, both of which is a 94% reduction for their respective problems. The global behavior of the action-value function was lost during learning at finer levels of discretization. As such, the agent autonomously uses the approximations at the end of each level of discretization when appropriate. Finally, Q-learning with policy comparison and the genetic algorithm for function approximation was demonstrated with the 3-dimensional multiple thermal case. Results here show that the action-value function is approximated and only 600 episodes are needed, which is an 88% reduction, to converge to a usable policy extracted from the approximated action-value function.

CHAPTER XI

CONCLUSIONS

This dissertation describes the development and application of a Q-learning based algorithm intended to more efficiently learn complex state-spaces with particular attention paid to regions of interest containing high concentrations of information. The algorithm consists of three novel components in addition to the Q-learning algorithm. These three components are the multi-resolution state-space discretization method (AAG), policy comparison and performance stopping criteria (PC), and the genetic algorithm for function approximation. All three components working together form the MGAP algorithm.

To analyze the performance and usefulness of the algorithm, several problems are cast as reinforcement learning problems and learning conducted using the algorithm. The algorithm is applied to the inverted pendulum, which is a simple benchmark dynamic system. To represent the class of systems of particular interest in this dissertation, the morphing airfoil and morphing wing are both cast as reinforcement learning problems with the shape parameters as the state variables. Also, a thermal location for autonomous soaring problem is cast as a reinforcement learning problem to test the algorithm's ability to efficiently handle multiple regions of interest.

A series of simulations are performed for each problem to test the learning performance of the various components individually and in conjunction with each other. The results are analyzed in a variety of ways including a dimensionality analysis, Monte Carlo simulations, value function and policy analysis, and action-value approximation analysis when appropriate.

The following conclusions are made based on the results presented in this dissertation:

1. The multi-resolution state-space discretization method offered a good solution to the integration of the learning on a coarse discretization for fast convergence over the whole state-space with learning on a fine discretization for the capturing of fine detail in regions of interest. Conceptually, a problem with the full state-space discretized at the finest level would take many thousands or tens of thousands of episodes more to reach this level of convergence. This method is successful in greatly reducing the time for convergence, increasing the rate of convergence, and achieving a goal with a very small range. This method essentially reduced the larger problems by at least an order of magnitude, making each a much more tractable learning problem.

2. Convergence is sensitive to state-space discretization. If the discretization is very fine, then rate of convergence is slower, taking upwards of thousands of episodes to achieve $> 98\%$ success in the case of the morphing airfoil. When the discretization is coarse, only hundreds of episodes are needed to achieve $> 98\%$ success. Often, only 200 to 400 episodes were necessary for the action-value function of a coarse discretization to converge to a policy that has $> 98\%$ success. However, a fine discretization is desired for a refined goal or cost function.

3. The policy comparison, when integrated into Q-learning with AAG, showed a significant reduction in the number of episodes necessary. Based on the user defined allowed episodes per level of discretization, this reduction ranged from 28% to 93.7% for the various problems.

4. Integrating the genetic algorithm into the algorithm allows for the approximation of the action-value function to evolve with the function itself. In many of the cases, the total number of weights, knots, and/or centers is a fraction of the

total number of state-action pairs. The exceptions are the wing and last thermal location problems in which the approximations are also large and available computational ability becomes an issue.

CHAPTER XII

RECOMMENDATIONS

Several recommendations are made here based on the research presented in this dissertation:

1. Investigate the effect of uncertainty on the state of the environment as sensed by the agent. All of the reinforcement learning problems in this dissertation assume perfect knowledge of the current state of the environment, which is not representative of the true environment. This may be done by adding noise to the state, $s \in S$. The Q-learning update law can then be applied to the *quantized* state and stored in the action-value function, similar in essence to how Q-learning was applied to the inverted pendulum reinforcement learning problem.

2. Explore the effect of incorporating transition probabilities or conditional probabilities. It is assumed that given a state and action, the next occurs with a probability of 1. This is not the case in many applications. Instead, the probability of transitioning from one state to another can be governed by a Gaussian probability density function, an exponential density function, *etc.*

3. Extend the multi-resolution state-space discretization method to discretizing the state-space by randomly distributing the states. This randomization will necessitate changes in the action definitions, but it will further reduce the effects of dimensionality.

4. Integrate a more sophisticated function approximator, such as GLOMAP, into the genetic algorithm in place of the fitness function. GLOMAP should be able

to capture the global behavior of the action-value function as well as the local behavior in the areas that are rediscretized, which the simple application of linear least-squares could not do necessitating the need for approximations of the action-value function after learning for each level of discretization. This could also alleviate the computation memory issues encountered in the larger problems when the genetic algorithm is applied.

5. Investigate reducing the range of the second gene of the genetic algorithm. The morphing wing example shows that good approximations are achieved when this gene is limited to 6 or less. Additionally, investigate the effects of modifying the genetic algorithm in such a way that members of the population with lower values for the second gene that are good approximations are favored over members with higher values for the second gene that are as good or only marginally better.

6. Investigating penalizing higher order terms of the basis functions using noise characteristics.

7. Investigate randomizing or intelligently distributing the nodes, centers, and knots in the genetic algorithm. Uniform distribution encounters similar dimensionality problems as rigidly discretized state-spaces. Randomizing the distribution should alleviate this problem in the genetic algorithm.

8. The current incarnation of the thermal location problem identifies the center and path to the nearest thermal from any point in the state-space. It would be beneficial to identify the center of the thermals at the coarsest discretization and then learn the path to each thermal separately while avoiding the rest of the thermals.

9. Extend the multi-resolution state-space discretization method to note Regions Of Interest that are "bad" or are difficult regions or boundaries. Rediscretizing the area and continuing learning in that region will refine the path around the region to avoid it, escape it, or locate a safe path through the region.

10. Further develop the algorithm to apply to problems in which a single action can change one or more state variables. The problems presented in this dissertation limited the number of admissible actions and assumed that taking an action changes only one state. This is not the case for many applications. This algorithm could benefit these applications in a similar manner as those presented in this dissertation.

11. Explore methods for choosing what order to evaluate multiple Regions Of Interest. Currently the order in which subsequent learning is conducted on multiple Regions Of Interest is random. Conducting subsequent learning by order of which region has the most information or some other measure would better direct the agent.

12. Develop performance guarantees for the approach. As other methods have done, bounds can be set at the boundary with linear point designs to guarantee this approach with a traditional and well understood approach.

13. Investigate alternative reinforcement learning problem definitions for the applications presented in this dissertation. There will be issues in the future in getting this kind of configuration controller certified on a reconfigurable aircraft. Therefore, it may be beneficial applying a more traditional controller and having the approach developed here learn how to tune or change gains to meet performance requirements throughout the flight envelope.

REFERENCES

[1] A. K. Jha and J. N. Kudva, "Morphing aircraft concepts, classifications, and challenges," in *Proc. of SPIE - Industrial and Commercial Applications of Smart Structures Technologies*, San Diego, CA, 16-18 March 2004, pp. 213–224.

[2] C. E. S. Cesnik, H. R. Last, and C. A. Martin, "A framework for morphing capability assessment," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1654.

[3] J. Bowman, "Affordability comparison of current and adaptive and multi-functional air vehicle systems," in *44th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference and Exhibit*, Norfold, VA, 7-10 April 2003, number AIAA-2003-1713.

[4] D. N. Talley, N. Schellpfeffer, C. Johnson, and D. N. Mavris, "Methodology for the mission requirement determination and conceptual design of a morphing UCAV," in *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, Chicago, IL, 20-23 September 2004, number AIAA-2004-6597.

[5] D. N. Talley and D. N. Mavris, "An adaptive environment for the identification of morphing UCAV mission requirements," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 9-12 January 2006, number AIAA-2006-822.

[6] C. Peters, B. Roth, W. A. Crossley, and T. A. Weisshaar, "Use of design methods to generate and develop missions for morphing aircraft," in *9th*

AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, GA, 4-6 September 2002, number AIAA-2002-5468.

[7] J. B. Frommer and W. A. Crossley, "Evaluating morphing aircraft in a fleet context using non-deterministic metrics," in *AIAA 5th Aviation, Technology, Integration, and Operations Conference (ATIO)*, Arlington, VA, 26-28 September 2005, number AIAA-2005-7460.

[8] J. B. Frommer and W. A. Crossley, "Building surrogate models for capability-based evaluation: Comparing morphing and fixed geometry aircraft in a fleet context," in *6th AIAA Aviation Technology, Integration and Operations Conference (ATIO)*, Wichita, KS, 25-27 September 2006, number AIAA-2006-7700.

[9] J. C. Bowman, R. W. Plumley, J. A. Dubois, and D. M. Wright, "Mission effectiveness comparisons of morphing and non-morphing vehicles," in *6th AIAA Aviation Technology, Integration and Operations Conference (ATIO)*, Wichita, KS, 25-27 September 2006, number AIAA-2006-7771.

[10] E. T. Martin and W. A. Crossley, "Multiobjective aircraft design to investigate potential geometric morphing features," in *AIAA's Aircraft Technology, Integration, and Operations (ATIO) 2002 Technical Conference*, Los Angeles, CA, 1-3 October 2002, number AIAA-2002-5859.

[11] S. E. Gano, V. M. Perez, J. E. Renaud, S. M. Batill, and B. Sanders, "Multilevel variable fidelity optimization of a morphing unmanned aerial vehicle," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1763.

[12] S. Ricci and M. Terraneo, "Application of mdo techniques to the preliminary design of morphed aircraft," in *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Portsmouth, VA, 6-8 September 2006, number AIAA-2006-7018.

[13] I. Ordaz, K. H. Lee, D. M. Clark, and D. N. Mavris, "Aerodynamic optimization using physics-based response surface methodology for a multi-mission morphing unmanned combat air vehicle," in *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, Chicago, IL, 20-23 September 2004, number AIAA-2004-6336.

[14] B. Roth, C. Peters, and W. A. Crossley, "Aircraft sizing with morphing as an independent variable: Motivation, strategies, and investigations," in *AIAA's Aircraft Technology, Integration, and Operations (ATIO) 2002 Technical*, Los Angeles, CA, 1-3 October 2002, number AIAA-2002-5840.

[15] B. D. Roth and W. A. Crossley, "Application of optimization techniques in the conceptual design of morphing aircraft," in *AIAA's 3rd Annual Aviation Technology, Integration, and Operations (ATIO) Technical Conference*, Denver, CO, 17-19 November 2003, number AIAA-2003-6733.

[16] J. B. Frommer and W. A. Crossley, "Enabling continuous optimization for sizing morphing aircraft concepts," in *43rd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 10-13 January 2005, number AIAA-2005-816.

[17] M. D. Skillen and W. A. Crossley, "Developing response surface based wing weight equations for conceptual morphing aircraft sizing," in *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials*

Conference and Exhibit, Austin, TX, 18-21 April 2005, number AIAA-2005-1960.

[18] M. D. Skillen and W. A. Crossley, "Developing morphing wing weight predictors with emphasis on the actuation mechanism," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-2042.

[19] C. H. Hong, M. Cheplak, J. Choi, and D. N. Mavris, "Flexible multi-body design of a morphing UCAV," in *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, Chicago, IL, 20-23 September 2004, number AIAA-2004-6595.

[20] J. M. Lafleur, J. R.Olds, and R. D. Braun, "Daedalon: A revolutionary morphing spacecraft design for planetary exploration," in *1st Space Exploration Conference: Continuing the Voyage of Discovery*, Orlando, FL, 30 January - 1 February 2005, number AIAA-2005-2771.

[21] J. G. Nehrbass, J. B. Frommer, L. A. Garison, D. N. Loffing, and W. A. Crossley, "Point to point commercial aircraft service design study including formation flight and morphing," in *AIAA 4th Aviation Technology, Integration and Operations (ATIO) Forum*, Chicago, IL, 20-22 September 2004, number AIAA-2004-6404.

[22] J. Cistone, "Next century aerospace traffic management: The sky is no longer the limit," *Journal of Aircraft*, vol. 41, no. 1, pp. 36–42, January-February 2004.

[23] R. K. Nangia and M. E. Palmer, "Morphing UCAV wings incorporating in-plane & folded-tips – aerodynamic design studies," in *24th Applied Aerodynamics Conference*, San Francisco, CA, 5-8 June 2006, number AIAA-2006-2835.

[24] J. Vale, F. Lau, A. Suleman, and P. Gamboa, "Multidisciplinary design optimization of a morphing wing for an experimental UAV," in *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Portsmouth, VA, 6-8 September 2006, number AIAA-2006-7131.

[25] J. Blondeau, J. Richeson, and D. J. Pines, "Design, development, and testing of a morphing aspect ratio wing using an inflatable telescopic spar," in *44th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference and Exhibit*, Norfold, VA, 7-10 April 2003, number AIAA-2003-1718.

[26] J. E. Blondeau and D. J. Pines, "Pneumatic morphing aspect ratio wing," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1808.

[27] D. Cadogan, T. Smith, F. Uhelsky, and M. MacKusick, "Morphing inflatable development for compact package unmanned aerial vehicles," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1807.

[28] S. E. Gano and J. E. Renaud, "Optimized unmanned aerial vehicle with wing morphing for extended range and endurance," in *9th AIAA/ISSMO Symposium*

on Multidisciplinary Analysis and Optimization, Atlanta, GA, 4-6 September 2002, number AIAA-2002-5668.

[29] M. B. Webb and K. Subbarao, "On the dynamic aeroelastic stability of morphable wing stuctures," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-2133.

[30] J. Bae, T. M. Seigler, and D. J. Inman, "Aerodynamic and static aeroelastic characteristics of a variable-span morphing wing," *Journal of Aircraft*, vol. 42, no. 2, pp. 528–534, March-April 2005.

[31] F. H. Gern, D. J. Inman, and R. K. Kapania, "Structural and aeroelastic modeling of general planform wings with morphing airfoils," *AIAA Journal*, vol. 40, no. 4, pp. 628–637, April 2002.

[32] J. N. Scarlett, R. A. Canfield, and B. Sanders, "Multibody dynamic aeroelastic simulation of a folding wing aircraft," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-2135.

[33] M. H. Love, P. S. Zink, R. L. Stroud, D. R. Bye, and C. Chase, "Impact of actuation concepts on morphing aircraft structures," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1724.

[34] J. J. Granda, I. Q. Sandoval, and L. G. Horta, "Morphing structural concepts evaluation criteria using dimensionless analysis and computer simulation," in

46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit, Austin, TX, 18-21 April 2005, number AIAA-2005-2111.

[35] L. D. Wiggins, M. D. Stubbs, C. O. Johnston, H. H. Robertshaw, C. F. Reinholtz, and D. J. Inman, "A design and analysis of a morphing hyper-elliptic cambered span (hecs) wing," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1885.

[36] D. S. Ramrakhyani, G. A. Lesieutre, M. Frecker, and S. Bharti, "Aircraft structural morphing using tendon-actuated compliant cellular trusses," *Journal of Aircraft*, vol. 42, no. 6, pp. 1615–1621, November-December 2005.

[37] M. R. Schultz, "A concept for airfoil-like active bistable twisting structures," *Journal of Intelligent Material Systems and Structures*, vol. 19, pp. 157–169, February 2008.

[38] D. A. Perkins, J. L. Reed Jr., and E. Havens, "Morphing wing structures for loitering air vehicles," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1888.

[39] K. Lu and S. Kota, "Parameterization strategy for opimization of shape morphing compliant mechanisms using load path representation," in *ASME Design Engineering Techinical Conference*, Chicago, IL, 2-6 September 2003, number DETC2003/DAC-48775, pp. 693–702.

[40] K. Lu and S. Kota, "An effective method of synthesizing compliant adaptive

structures using load path representation," *Journal of Intelligent Material Systems and Structures*, vol. 16, pp. 307–317, April 2005.

[41] P. Vu, F. Chavez, A. Kelkar, and C. Whitmer, "Investigation of the effects of stiffness on control power via a morphing wing technology," in *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Austin, TX, 18-21 April 2005, number AIAA-2005-2039.

[42] K. Boothe, "A design scheme aimed at minimizing actuator and structural loads of a span morphing aircraft," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-2137.

[43] F. H. Gern, D. J. Inman, and R. K. Kapania, "Computation of actuation power requirements for smart wings with morphing airfoils," *AIAA Journal*, vol. 43, no. 12, pp. 2481–2486, April 2005.

[44] J. H. Mabe, F. R. Calkins, and G. W. Butler, "Boeing's variable geometry chevron morphing aerostructure for jet noise reduction," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-2142.

[45] G. S. Bushnell, D. Arbogast, and R. Ruggeri, "Shape control of a morphing structure (rotor blade) using a shape memory alloy actuator system," in *Proc. of SPIE - Active Passive Smart Structures and Integrated Systems*, San Diego, CA, 10-13 March 2008.

[46] S. P. Joshi, Z. Tidwell, W. A. Crossley, and S. Ramakrishnan, "Comparison of morphing wing strategies based upon aircraft perfromance impacts," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1722.

[47] M. T. Rusnell, S. E. Gano, V. M. Perez, J. E. Renaud, and S. M. Batill, "Morphing UAV pareto curve shift for enhanced performance," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1682.

[48] G. W. Reich, J. C. Bowman, B. Sanders, and G. J. Frank, "Development of an integrated aeroelastic multibody morphing simulation tool," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-1892.

[49] T. M. Seigler, D. A. Neal, and D. J. Inman, "Dynamic modeling of large-scale morphing aircraft," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-1893.

[50] Jason C. Bowman, Gregory W. Reich, Brian Sanders, and Geoffrey J. Frank, "Simulation tool for analyzing complex shape-changing mechanisms in aircraft," in *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Keystone, CO, 21-24 August 2006, number AIAA-2006-6727.

[51] E. Cuji and E. Garcia, "Prediction of aircraft dynamics with shape changing

wings," in *Proc. of SPIE - Active Passive Smart Structures and Integrated Systems*, San Diego, CA, 10-13 March 2008.

[52] J. B. Davidson, P. Chwalowski, and B. S. Lazos, "Flight dynamic simulation assessment of a morphable hyper-elliptic cambered span winged configuration," in *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Austin, TX, 11-14 August 2003, number AIAA-2003-5301.

[53] M. A. Scott, R. C. Montgomery, and R. P. Weston, "Subsonic maneuvering effectiveness of high performance aircraft which employ quasi-static shape change devices," in *Proc. of the SPIE - 5th Annual International Symposium on Structures and Materials*, San Diego, CA, 1-6 March 1998, pp. 223–233.

[54] D. H. Baldelli, D. Lee, R. S. S. Pena, and B. Cannon, "Modeling and control of an aeroelastic morphing vehicle," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 6, pp. 1687–1699, November-December 2008.

[55] C. E. Whitmer and A. G. Kelkar, "Robust control of a morphing airfoil structure," in *2005 American Control Conference*, Portland, OR, 8-10 June 2005, pp. 2863–2868.

[56] A. Ataie-Esfahani and Q. Wang, "Robust failure compensation for a morphing aircraft model using a probabilistic approach," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 2, pp. 324–331, March 2007.

[57] G. Tao, X. Tang, S. Chen, J. Fei, and S. M. Joshi, "Adaptive failure compensation of two-state aircraft morphing actuators," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 1, pp. 157–164, January 2006.

[58] B. Sanders, F. E. Eastep, and E. Forster, "Aerodynamic and aeroelastic characteristics of wings with conformal control surfaces for morphing aircraft," *Journal of Aircraft*, vol. 40, no. 1, pp. 94–99, January-February 2003.

[59] C. O. Johnston, D. A. Neal, L. D. Wiggins, H. H. Robertshaw, W. H. Mason, and D. J. Inman, "A model to compare the flight control energy requirements of morphing and conventionally actuated wings," in *44th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Norfold, VA, 7-10 April 2003, number AIAA-2003-1716.

[60] J. Blondeau and D. J. Pines, "Wind tunnel testing of a morphing aspect ratio wing using a pneumatic telescopic spar," in *2nd AIAA "Unmanned Unlimited" Systems, Technologies, and Operations Aerospace*, San Diego, CA, 15-18 September 2003, number AIAA-2003-6659.

[61] D. A. Neal III, J. Farmer, and D. Inman, "Development of a morphing aircraft model for wind tunnel experimentation," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-2141.

[62] R. Guiler and W. Huebsch, "Wind tunnel analysis of a morphing swept wing tailless aircraft," in *23rd AIAA Applied Aerodynamics Conference*, Toronto, Ontario Canada, 6-9 June 2005, number AIAA-2005-4981.

[63] F. Boria, B. Stanford, W. Bowman, and P. Ifju, "Evolutionary optimization of a morphing wing with wind tunnel hardware-in-the-loop," in *47th AIAA*

Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition, Orlando, FL, 5-8 January 2009, number AIAA-2009-1460.

[64] J. E. Hubbard Jr, "Dynamic shape control of a morphing airfoil using spatially distributed transducers," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 29, no. 3, pp. 612–616, 2006.

[65] N. M. Ursache, A. J. Keane, and N. W. Bressloff, "Design of postbuckled spinal structures for airfoil camber and shape control," *AIAA Journal*, vol. 44, no. 12, pp. 3115–3124, December 2006.

[66] M. Secanell, A. Suleman, and P. Gamboa, "Design of a morphing airfoil using aerodynamic shape optimization," *AIAA Journal*, vol. 44, no. 7, pp. 1550–1562, July 2006.

[67] H. Namgoong, W. A. Crossley, and A. S. Lyrintzis, "Aerodynamic optimization of a morphing airfoil using energy as an objective," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 9-12 January 2006, number AIAA-2006-1324.

[68] H. Namgoong, W. A. Crossley, and A. S. Lyrintzis, "Morphing airfoil design for minimum aerodynamic drag and actuation energy including aerodynamic work," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Newport, RI, 1-4 May 2006, number AIAA-2006-2041.

[69] B. C. Prock, T. A. Weisshaar, and W. A. Crossley, "Evaluating the impact of morphing technologies on aircraft performance," in *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA, 4-6 September 2002, number AIAA-2002-5401.

[70] H. M. Garcia, M. Abdulrahim, and R. Lind, "Roll control for a micro air vehicle using active wing morphing," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, TX, 11-14 August 2003, number AIAA-2003-5347.

[71] M. Abdulrahim, H. M. Garcia, G. F. Ivey, and R. Lind, "Flight testing a micro air vehicle using morphing for aeroservoelastic control," in *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, Palm Springs, CA, 19-22 April 2004, number AIAA-2004-1674, pp. 1776–1792.

[72] B. Stanford, M. Abdulrahim, R. Lind, and P. Ifju, "Investigation of membrane actuation for roll control of a micro air vehicle," *Journal of Aircraft*, vol. 44, no. 3, pp. 741–749, May-June 2007.

[73] M. Abdulrahim, H. M. Garcia, and R. Lind, "Flight characteristics of shaping the membrane wing of a micro air vehicle," *Journal of Aircraft*, vol. 42, no. 1, pp. 131–137, January-February 2005.

[74] M. Abdulrahim and R. Lind, "Flight testing and response characteristics of a variable gull-wing morphing aircraft," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, RI, 16-19 August 2004, number AIAA-2004-5113, pp. 1664–1679.

[75] M. Abdulrahim, "Flight performance characteristics of a biologically-inspired morphing aircraft," in *AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 10-13 January 2005, number AIAA-2005-345, pp. 6657–6671.

[76] M. Abdulrahim and R. Lind, "Using avian morphology to enhance aircraft

maneuverability," in *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Keystone, CO, 21-24 August 2006, number AIAA-2006-6643.

[77] D. T. Grant, M. Abdulrahim, and R. Lind, "Flight dynamics of a morphing aircraft utilizing independent multiple-joint wing sweep," in *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Keystone, CO, 21-24 August 2006, number AIAA-2006-6505, pp. 1111–1125.

[78] R. Albertani, B. Stanford, J. P. Hubner, R. Lind, and P. Ifju, "Experimental analysis of deformation for flexible-wing micro air vehicles," in *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, Austin, TX, 18-21 April 2005, number AIAA-2005-2231, pp. 5339–5352.

[79] K. Boothe, K. Fitzpatrick, and R. Lind, "Controllers for disturbance rejection for a linear input-varying class of morphing aircraft," in *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, Austin, TX, 18-21 April 2005, number AIAA-2005-2374.

[80] M. Abdulrahim and R. Lind, "Control and simulation of a multi-role morphing micro air vehicle," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, CA, 15-18 August 2005, number AIAA-2005-6481.

[81] R. Sutton and A. Barto, *Reinforcement Learning - An Introduction*, pp. 9, 51–80, 89–100, 111, 133, 148–150, Cambridge, Massachusetts, The MIT Press, 1998.

[82] Dimitri P. Bertsekas, "Dynamic programming and suboptimal control: A survey from ADP to MPC," *European Journal of Control*, vol. 11, no. 4, pp. 310–334, April 2005.

[83] D. P. Bertsekas, "Distributed dynamic programming," *IEEE Transactions on Automatic Control*, vol. AC-27, no. 3, pp. 610–616, June 1982.

[84] D. P. Bertsekas, "Separable dynamic programming and approximate decomposition methods," *IEEE Transactions on Automatic Control*, vol. 52, no. 5, pp. 911–916, May 2007.

[85] D. H. Jacobson, "Differential dynamic programming methods for solving bang-bang control problems," *IEEE Transactions on Automatic Control*, vol. AC-13, no. 6, pp. 661–675, December 1968.

[86] S. B. Gershwin and D. H. Jacobson, "A discrete-time differential dynamic programming algorithm with application to optimal orbit transfer," *AIAA Journal*, vol. 8, no. 9, pp. 1616–1626, September 1970.

[87] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-dynamic programming: An overview," in *Proc. of the 34th IEEE Conference on Decision & Control*, New Orleans, LA, December 1995, pp. 560–564.

[88] S. N. Balakrishnan, J. Ding, and F. L. Lewis, "Issues on stability of ADP feedback controllers for dynamical systems," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 913–917, August 2008.

[89] S. Ferrari, J. E. Steck, and R. Chandramohan, "Adaptive feedback control by constrained approximate dynamic programming," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 982–987, August 2008.

[90] G. G. Lendaris, "Higher level application of ADP: A next phase for the control field?," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 901–912, August 2008.

[91] D. P. Bertsekas, "New value iteration and q-learning methods for the average cost dynamic programming problem," in *Proc. of the 37th IEEE Conference on Decision & Control*, Tampa, FL, December 1998, pp. 2692–2697.

[92] C. G. Atkeson and B. J. Stephens, "Random sampling of states in dynamic programming," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 924–929, August 2008.

[93] C. Lu, J. Si, and X. Xie, "Direct heuristic dynamic programming for damping oscillations in a large power system," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 1008–1013, August 2008.

[94] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Artificial Intelligence Review*, vol. 11, pp. 75–113, 1997.

[95] K. L. Moore, M. Dahleh, and S. P. Bhattacharyya, "Iteratively learning for trajectory control," in *Proc. of the 28th Conference on Decision & Control*, Tampa, FL, December 1989, pp. 860–865.

[96] J. Hatonen, K. L. Moore, and D. H. Owens, "An algebraic approach to iterative learning control," in *Proc. of the 2002 IEEE International Symposium on Intelligent Control*, Vancouver, Canada, 27-30 October 2002, pp. 37–42.

[97] K. L. Moore H. Ahn and Y. Q. Chen, "Stability analysis of discrete-time iterative learning control systems with interval uncertainty," *Automatica*, vol. 43, pp. 892–902, 2007.

[98] C. Chien, "A combined adaptive law for fuzzy iterative learning control of nonlinear systems with varying control tasks," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 1, pp. 40–51, February 2008.

[99] P. Y. Glorennec, "Fuzzy q-learning and dynamical fuzzy q-learning," in *Proc. of the IEEE International Conference on Fuzzy Systems*, Orlanda, FL, 26-29 June 1994, pp. 474–479.

[100] H. R. Berenji, "Fuzzy q-learning for generalization of reinforcement learning," in *Proc. of the IEEE International Conference on Fuzzy Systems*, New Orleans, LA, 8-11 September 1996, pp. 2208–2214.

[101] Y. Maeda, "Modified q-learning method with fuzzy state division and adaptive rewards," in *Proc. of the 2002 IEEE International Conference on Fuzzy Systems*, Honolulu, HI, 12-17 May 2002, pp. 1556–1561.

[102] L. Busoniu, D. Ernst, B. D. Schutter, and R. Babuska, *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, chapter Continuous-State Reinforcement Learning with Fuzzy Approximation, pp. 27–43, Berlin, Germany, Springer, 2008.

[103] K. Takahashi, H. Ueda, and T. Miyahara, "Agent learning in simulated soccer by fuzzy q-learning," in *IEEE TENCON 2004 - 2004 IEEE Region 10 Conference Proceedings: Analog and Digital Techniques in Electrical Engineering*, Chiang Mai, Thailand, 21-24 November 2004.

[104] R. Sharma and M. Gopal, "Hybrid game strategy in fuzzy markov game based control," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 5, pp. 1315–1327, May 2008.

[105] R. Sharma and M. Gopal, "A markov game-adaptive fuzzy controller for robot manipulators," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 1, pp. 171–186, February 2008.

[106] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *Proc. of the 23rd International Conference on Machine Learning*, Pittsburg, PA, 25-29 June 2006, pp. 1–8.

[107] T. J. Perkins and A. G. Barto, "Lyapunov design for safe reinforcement learning," *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 803–832, May 2003.

[108] A. Tan, N. Lu, and D. Xiao, "Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 230–244, February 2008.

[109] M. A. Wiering and H. van Hasselt, "Ensemble algorithms in reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 930–936, August 2008.

[110] C. Chen, P. Yang, X. Zhou, and D. Dong, "A quantum-inspired q-learning algorithm for indoor robot navigation," in *Proc. of the IEEE International Conference on Networking, Sensing and Control*, Sanya, China, 6-8 April 2008, pp. 1599–1603.

[111] D. Dong, C. Chen, T. Tarn, A. Pechen, and H. Rabitz, "Incoherent control of quantum systems with wavefunction-controllable subspaces via quantum reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 957–962, August 2008.

[112] L. Busoniu, R. Babuska, and B. D. Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*, vol. 38, no. 2, pp. 156–172, March 2008.

[113] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *Proc. of the 19th International Conference on Machine Learning*, Sydney, Australia, July 2002, pp. 227–234.

[114] T. Liang and L. Ji-lian, "Multi-agent reinforcement learning algorithm based on action prediction," *Journal of Beijing Institute of Technology*, vol. 15, no. 2, pp. 133–137, June 2006.

[115] D. C. Bentivegna, C. G. Atkeson, and G. Cheng, "Learning to select primitives and generate sub-goals from practice," in *Proc. of the 2003 IEEE/lRSJ Intl. Conference on Intelligent Robots and Systems*, Las Vegas, NV, October 2003, pp. 946–953.

[116] O. Simsek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in *Proc. of the 22nd International Conference on Machine Learning*, Bonn, Germany, 7-11 August 2005, pp. 817–824.

[117] C. Clausen and H. Wechsler, "Quad-q-learning," *IEEE Transactions on Neural Networks*, vol. 11, no. 2, pp. 279–294, February 2000.

[118] G. Theocharous and S. Mahadevan, "Learning the hierarchical structure of spatial environments using multiresolution statistical models," in *Proc. of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, October 2002, pp. 1038–1043.

[119] T. G. Dietterich, "The maxq method for hierarchical reinforcement learning," in *Proc. of the Fifteenth International Conference on Machine Learning*, Madison, WI, 24-27 July 1998, pp. 118–126.

[120] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, November 2000.

[121] C. Diuk, A. L. Strehl, and M. L. Littman, "A hierarchical approach to efficient reinforcement learning in deterministic domains," in *AAMAS'06*, Hakodate, Hokkaido, Japan, 8-12 May 2006.

[122] S. Elfwing, E. Uchibe, K. Doya, and H. I. Christensen, "Evolutionary development of hierarchical learning structures," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 249–264, April 2007.

[123] R. Makar, S. Mahadevan, and M. Ghavamzadeh, "Hierarchical multi-agent reinforcement learning," in *AGENTS'01*, Montreal, Quebec, Canada, 28 May-1 June 2001.

[124] F. Kirchner, "Q-learning of complex behaviours on a six-legged walking machine," *Robotics and Autonomous Systems*, vol. 25, no. 4, pp. 253–262, November 1998.

[125] C. Chen, H. Li, and D. Dong, "Hybrid control for robot navigation: A hierarchical q-learning algorithm," *IEEE Robotics & Automation Magazine*, pp. 37–47, June 2008.

[126] M. Ghavamzadeh and S. Mahadevan, "Learning to communicate and act using hierarchical reinforcement learning," in *Proc. of the Third International Joint*

Conference on Autonomous Agents and Multiagent Systems, New York,NY, 19-23 July 2004, pp. 1114–1121.

[127] C. M. Vigorito and A. G. Barto, "Hierarchical representations of behavior for efficient creative search," in *AAAI Spring Symposium - Technical Report*, Stanford, CA, 26-28 May 2008, pp. 135–141.

[128] B. Baddeley, "Reinforcement learning in continuous time and space: Interference and not ill conditioning is the main problem when using distributed function approximators," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 950–956, August 2008.

[129] C. W. Phua and R. Fitch, "Tracking value function dynamics to improve reinforcement learning with piecewise linear function approximation," in *Proc. of the 24th International Conference on Machine learning*, Corvalis, OR, 20-24 July 2007, pp. 751–758.

[130] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, "Efficient solution algorithms for factored mdps," *Journal of Artificial Intelligence Research*, vol. 19, pp. 399–468, July/December 2003.

[131] L. C. Baird III, "Reinforcement learning in continuous time: Advantage updating," in *Proc. of the 1994 IEEE International Conference on Neural Networks*, Orlando, FL, 27-29 June 1994, pp. 2448–2453.

[132] S. H. Kim, I. H. Suh, S. R. Oh, Y. J. Cho, and Y. K. Chung, "Region-based q-learning using clustering approach," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robot and Systems*, Grenoble, France, 7-11 September 1998, pp. 601–607.

[133] T. Fukao, T. Sumitomo, N. Ineyama, and N. Adachi, "Q-learning based on regularization theory to treat the continuous states and actions," in *Proc. of the 1998 IEEE International Joint Conference on Neural Networks*, Anchorage, AK, 4-9 May 1998, pp. 1057–1062.

[134] L. Hu, C. Zhou, and Z. Sun, "Estimating biped gait using spline-based probability distribution function with q-learning," *IEEE Transactionson Industrial Electronics*, vol. 55, no. 3, pp. 1444–1452, March 2008.

[135] I. H. Suh, J. H. Kim, and S. R. Oh, "Region-based q-learning for intelligent robot systems," in *Proc. of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Monterey, CA, 10-11 July 1997, pp. 172–178.

[136] Y. Takahashi, M. Takeda, and M. Asada, "Continuous valued q-learning for vision-guided behavior acquisition," in *Proc. of the 1999 IECE International Conlerencc on Multisensor Fusion and Integration for Intelligent Systems*, Taipei, Taiwan, August 1999, pp. 255–260.

[137] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," *Machine Learning*, vol. 22, no. 1-3, pp. 33–57, January-March 1996.

[138] A. Nedic and D. P. Bertsekas, "Least squares policy evaluation algorithms with linear function approximation," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 13, no. 1-2, pp. 79–110, January/April 2003.

[139] D. P. Bertsekas, A. Nedich, and V. S. Borkar, chapter Improved Temporal Difference Methods with Linear Function Approximation, in *Learning and Ap-*

proximate Dynamic Programming, pp. 235–259, Piscataway, New Jersey, Wiley-Interscience, 2004.

[140] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning*, vol. 4, no. 6, pp. 227–303, August 2004.

[141] M. G. Lagoudakis, R. Parr, and M. L. Littman, chapter Least-Squares Methods in Reinforcement Learning for Control, in *Methods and Applications of Artificial Intelligence*, pp. 249–260, Berlin, Germany, Springer, 2002.

[142] Y. B. Reddy, "Detecting primary signals for efficient utilization of spectrum using q-learning," in *Proc. of the International Conference on Information Technology: New Generations*, Las Vegas, NV, 7-9 April 2008, pp. 360–365.

[143] D. Shiqiang L. Shoufeng, L. Ximin, "Q-learning for adaptive traffic signal control based on delay minimization strategy," in *Proc. of 2008 IEEE International Conference on Networking, Sensing and Control*, Sanya, China, 6-8 April 2008, pp. 687–691.

[144] J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," in *Proc. of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004, pp. 3030–3034.

[145] S. Schaal and C. Atkeson, "Robot juggling: Implementation of memory-based learning," *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 57–71, February 1994.

[146] D. C. Bentivegna, C. G. Atkeson, and G. Cheng, "Learning tasks from observation and practice," *Robotics and Autonomous Systems*, vol. 47, pp. 163–169,

2004.

[147] S. Ray, G. K. Venayagamoorthy, B. Chaudhuri, and R. Majumder, "Comparison of adaptive critic-based and classical wide-area controllers for power systems," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 1002–1007, August 2008.

[148] D. Liu, H. Javaherian, O. Kovalenko, and T. Huang, "Adaptive critic learning techniques for engine torque and airfuel ratio control," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 38, no. 4, pp. 988–993, August 2008.

[149] J. Wharington, "Autonomous control of soaring aircraft by reinforcement learning," Ph.D. dissertation, Royal Melbourne Institute of Technology, Melbourne, Australia, 1998.

[150] E. van Kampen, Q. P. Chu, and J. A. Mulder, "Continuous adaptive critic flight control aided with approximated plant dynamics," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, CO, 21-24 August 2006, number AIAA-2006-6429.

[151] S. Bieniawski and I. M. Kroo, "Flutter suppression using micro-trailing edge effectors," in *44th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference*, Norfolk, VA, 7-10 April 2003, number AIAA-2001-1941.

[152] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for control from multiple demonstrations," in *Proc. of the 25th International Conference on Machine Learning*, Helsinki, Finland, 5-9 July 2008, pp. 144–151.

[153] J. A. Bagnell and J. G. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proc. of the 2001 IEEE International Conference on Robotics & Automation*, Seoul, Korea, 21-26 May 2001, pp. 1615–1620.

[154] R. Enns and J. Si, "Apache helicopter stabilization using neural dynamic programming," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 19–25, January/February 2002.

[155] S. Goel and P. Hajela, "Turbine aerodynamic design using reinforcement learning based optimization," in *AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, 2-4 September 1998, number AIAA-1998-4774, pp. 528–543.

[156] M. Motamed and J. Yan, "A reinforcement learning approach to lift generation in flapping mavs: Simulation results," in *Proc. of the 2006 IEEE International Conference on Robotics and Automation*, Orlando, FL, May 2006, pp. 2150–2155.

[157] M. Motamed and J. Yan, "A reinforcement learning approach to lift generation in flapping mavs: Experimental results," in *Proc. of the 2007 IEEE International Conference on Robotics and Automation*, Roma, Italy, 10-14 April 2007, pp. 748–754.

[158] J. Valasek, M. Tandale, and J. Rong, "A reinforcement learning - adaptive control architecture for morphing," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 4, pp. 174–195, April 2005.

[159] M. Tandale, J. Rong, and J. Valasek, "Preliminary results of adaptive-reinforcement learning control for morphing aircraft," in *AIAA Guidance,*

Navigation, and Control Conference and Exhibit, Providence, RI, 16-19 August 2004, number AIAA-2004-5358, pp. 3215–3225.

[160] J. Valasek, J. Doebbler, M. Tandale, and A. Meade, "Improved adaptive-reinforcement learning control for morphing unmanned air vehicles," *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, vol. 38, no. 4, pp. 1014–1020, August 2008.

[161] C. J. C. H. Watkins and P. Dayan, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, Cambridge, UK, 1989.

[162] P. Singla, "Multi-resolution methods for high fidelity modeling and control allocation in large-scale dynamical systems," Ph.D. dissertation, Texas A&M University, College Station, TX, 2006.

[163] H. Stark and J. W. Woods, *Probability and Random Processes with Applications to Signal Processing*, pp. 421–423, Upper Saddle River, NJ, Prentice Hall, 2002.

[164] T. M. Mitchell, *Machine Learning*, pp. 374–382, Boston, MA, The McGraw-Hill Companies, Inc., 1997.

[165] M. Guo, Y. Liu, and J. Malec, "A new q-learning algorithm based on the metropolis criterion," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, vol. 34, no. 5, pp. 2140–2143, October 2004.

[166] Q. Gao and B. Hong, "An improved q-learning algorithm based on exploration region expansion strategy," in *Proc. of the 6th World Congress on Intelligent Control and Automation*, Dalian, China, 21-23 June 2006, pp. 4167–4170.

[167] Y. Zheng, S. Luo, and J. Zhang, "Greedy exploration policy of q-learning based on state balance," in *Proc. of the IEEE Region 10 Annual International Conference*, Melbourne, Australia, 21-24 November 2005.

[168] A. Nowe, K. Steenhaut, M. Fakir, and K. Verbeeck, "Q-learning for adaptive, load based routing," in *Proc. of the 1998 IEEE International Conference on Systems, Man, and Cybernetics. Part 4*, San Diego, CA, 11-14 October 1998, pp. 3965–3970.

[169] M. A. Goodrich and M. Quigley, "Satisficing q-learning: Efficient learning in problems with dichotomous attributes," in *Proc. of the 2004 IEEE International Conference on Machine Learning and Applications*, Louisville, KY, 16-18 December 2004, pp. 65–72.

[170] O. Simsek and A. G. Barto, "An intrinsic reward mechanism for efficient exploration," in *ACM International Conference Proceeding Series*, Pittsburgh, PA, 25-29 June 2006, pp. 833–840.

[171] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvari, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 287–308, March 2000.

[172] A. Lampton, A. Niksch, and J. Valasek, "Reinforcement learning of a morphing airfoil-policy and discrete learning analysis," in *Proc. of the AIAA Guidance, Navigation, and Control Conference*, Honolulu, HI, 18-21 August 2008, number AIAA-2008-7281.

[173] A. J. Meade, M. Kokkolaras, and B. A Zeldin, "Sequential function approximation for the solution of differential equations," *Communications in Numerical Methods in Engineering*, vol. 13, no. 12, pp. 977–986, December 1997.

[174] M. Kokkolaras, A. J. Meade, and B. Zeldin, "Concurrent implementation of the optimal incremental approximation method for the adaptive and meshless solution of differential equations," *Optimization and Engineering*, vol. 4, no. 4, pp. 271–289, 2003.

[175] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman, "Analyzing feature generation for value-function approximation," in *Proc. of the 24th International Conference on Machine Learning*, Covallis, OR, 20-24 June 2007, pp. 737–744.

[176] Z. Lee, "A novel hybrid algorithm for function approximation," *Expert Systems with Applications*, vol. 34, no. 1, pp. 384–390, January 2008.

[177] J. W. Hauser, "Designing a genetic algorithm for function approximation for embedded and asic applications," in *Proc. of the 2006 49th Midwest Symposium on Circuits and Systems*, San Juan, Puerto Rico, 6-9 August 2006, pp. 555–559.

[178] J. L. Crassidis and J. L. Junkins, *Optimal Estimation of Dynamic Systems*, pp. 9–11, Boca Raton, FL, Chapman & Hall/CRC, 2004.

[179] S. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach*, pp. 624–625, Upper Saddle River, New Jersey, Pearson Education, Inc., 2003.

[180] Y. Zheng, S. Luo, and Z. Lv, "Control double inverted pendulum by reinforcement learning with double cmac network," in *Proc. of the 18th International Conference on Pattern Recognition*, Hong Kong, China, 20-24 August 2006, pp. 639–642.

[181] C.W. Tao, J. S. Taur, T. W. Hsieh, and C. L. Tsai, "Design of a fuzzy controller with fuzzy swing-up and parallel distributed pole assignment schemes for an

inverted pendulum and cart system," *IEEE Transactions on Control Systems Technology*, vol. 16, no. 6, pp. 1277–1288, November 2008.

[182] Y. Zheng, S. Luo, Z. Lv, and L. Wu, "Control double inverted pendulum by hierarchical reinforcement learning," in *Proc. of the 7th International Conference on Signal Processing Proceedings*, Beijing, China, August 31 - September 4 2004, pp. 1614–1617.

[183] A. Lampton, A. Niksch, and J. Valasek, "Reinforcement learning of morphing airfoils with aerodynamic and structural effects," *Journal of Aerospace Computing, Information, and Communication*, vol. 6, no. 1, pp. 30–50, January 2009.

[184] A. Niksch, J. Valasek, T. W. Strganac, and L. A. Carlson, "Morphing aircraft dynamical model: Longitudinal large scale shape changes," *Journal of Guidance, Control, and Dynamics*, (in review).

[185] M. J. Allen, "Updraft model for development of autonomous soaring uninhabited air vehicles," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 9-12 January 2006, number AIAA-2006-1510.

VITA

Amanda Kathryn Lampton graduated from Friendswood High School in Friendswood, TX in 2001. She subsequently attended Texas A&M University, where she received her Bachelor of Science in Aerospace Engineering (2004), Master of Science in Aerospace Engineering (2006), and Doctor of Philosophy in Aerospace Engineering (2009). Upon completion of her graduate degrees, she continues her search for meaningful employment. She may be contacted via the following address:

Texas A&M University
Aerospace Engineering Department
3141 TAMU
College Station, TX 77843-3141